UNCLASSIFIED

(2)

AD-A190 203

REPORT DOCUMENTATION PAGE

DTIC FILE COPY

1b. RESTRICTIVE MARKINGS	
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	3. DISTRIBUTION / AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S): AFOSR-TR- 87-1745
6a. NAME OF PERFORMING ORGANIZATION: Yale University; 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION: AFOSR/NM
6c. ADDRESS (City, State, and ZIP Code): New Haven, CT	7b. ADDRESS (City, State, and ZIP Code): AFOSR/NM Bldg 410 Bolling AFB DC 20332-6448
8a. NAME OF FUNDING / SPONSORING ORGANIZATION: AFOSR; 8b. OFFICE SYMBOL (If applicable): NM	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: F49620-82-K-0010
8c. ADDRESS (City, State, and ZIP Code): AFOSR/NM Bldg 410 Bolling AFB DC 20332-6448	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. 61102F; PROJECT NO. 2304; TASK NO. K1; WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Memory-Based Expert Systems

12. PERSONAL AUTHOR(S)
Professor Schank

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM 1/15/82 TO 9/29/85	June 1987	

16. SUPPLEMENTARY NOTATION

DTIC ELECTE JAN 14 1988 S D H

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

There are 3 major areas of accomplishments in recent AFOSR sponsored AI research at Yale.

While case-based resoning is simple in concept, there are of course many subtle and difficult design issues that have to be resolved to make it work. One major achievement in the past two years at Yale has been the development of an explicit structure for the case-based reasoning process, as shown in the slide containing the flowchart entitled, "Case-Based Reasoning." This process description summarizes results from the JUDGE (Bain 1986) AFOSR project as well as from other projects such as CHEF (Hammond 1986) and COACH (Collins forthcoming). The flow of control in the process (represented by the arrows and boxes) is task-independent, while the particular knowledge bases (represented by the ovals) are task specific. The JUDGE program models the subjective assessment task of judicial sentencing That is, given a description of an event, such as a fight between two people that ended in a death or serious injury, where someone has been convicted of a criminal act.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT: ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION
22a. NAME OF RESPONSIBLE INDIVIDUAL: Dr. Waksman	22b. TELEPHONE (Include Area Code): (202) 767-5027; 22c. OFFICE SYMBOL: NM

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

87 12 [illegible] 109

Major Research Efforts

There are three major areas of accomplishments in recent AFOSR sponsored AI research at Yale.

Case-based reasoning

Theoretical Background:

The current standard model of expert reasoning is rule-based. Expertise is encoded in hundreds to independent *if–then* rules. While this technique is an improvement over previous ad hoc coding methods, it has several major inadequacies as an approach to serious expertise:

- It is hard for programmers to extend and debug rules because they are not organized into a larger coherent structure.
- It is hard for domain experts to translate their expertise into rule form.
- It is hard for the program itself to learn, i.e., to automatically generate new rules from its own problem-solving experiences.

At Yale, we have developed an alternative to rule-based reasoning which we call ***case-based reasoning***. We believe case-based reasoning more accurately models human expertise and better supports learning and knowledge acquisition. The basic idea is simple: new problems are solved by adapting solutions to previously solved problems. A case-based reasoner is initialized with a case library of basic problems and solutions, which become the foundation of future problem-solving. The program adapts cases to new problems and adds successful adaptations to the library.

Such an approach has several promising advantages:

- Problem-solving behavior improves automatically, as solutions to common problems are saved for future use.
- Using simple substitution rules, the program can adapt and use a library solution that would be very difficult, if not impossible, to generate from scratch with a rule-based approach.
- Experts can extend the program simply by adding new example solutions to the library.

While case-based reasoning is simple in concept, there are of course many subtle and difficult design issues that have to be resolved to make it work. One major achievement in the past two years at Yale has been the development of an explicit structure for the case-based reasoning process, as shown in the slide containing the flowchart entitled "Case-Based Reasoning." This process description summarizes results from the JUDGE (Bain 1986) AFOSR project as well as from other projects such as CHEF (Hammond 1986) and COACH (Collins forthcoming). The flow of control in the process (represented by the arrows and boxes) is task-independent, while the particular knowledge bases (represented by the ovals) are task-specific.

There are two kinds of learning currently implemented in our programs, as indicated by arrows leading *into* knowledge bases. First, new solutions are stored in case memory for retrieval by future problem descriptions. This is "learning from success." Second, new rules for assigning indices to problem descriptions are stored whenever a retrieved solution fails to solve a problem. Such failures indicate that some feature in the problem description was overlooked. By checking for this feature in the future, inappropriate cases will not be retrieved. This is "learning from failure." Both kinds of learning are crucial to case-based reasoning.

This model is implemented in JUDGE, as well as several other problem-solvers. JUDGE deals with problems of subjective assessment. Subjective assessment is a kind of task that people have to do all the time, including appraisals of worth (houses, art, etc.,) judgments of merit (in gymnastics, the office, etc.,) and evaluations of goals (in self and others). In all subjective assessment problems, the explicitly known rules for assessment are very incomplete, and there is scant feedback regarding the accuracy of the assessment.

Case-based reasoning fits in very well here because it does not require a completely specified rule set. Instead it can use a small library of reasonable assessment solutions. In place of "correct/incorrect" feedback, JUDGE's adaptation rules work to maintain consistency with past assessments.

The JUDGE program models the subjective assessment task of judicial sentencing. That is, given a description of an event, such as a fight between two people that ended in a death or serious injury, where someone has been convicted of a criminal act, JUDGE determines a sentence that meets the rules of the law and is consistent with prior sentencing behavior. Each sentencing event is added to the case memory and used, when relevant, in later sentencing.

Explanation-based learning

An important extension of case-based reasoning is learning through explanation. We use the term explanation to refer to the construction of a causal model of an event or episode rather than to any particular communication a system might have with its users. Internal explanations are important so that a system can reason on the basis of the causal relationships it has discovered. Examples of tasks requiring this kind of explanation include adaptive recovery from failures in robot planning and navigation, diagnosis of faults in various hardware and software systems, and flexible behavior in the face of unexpected situations.

Furthermore, we believe that explanation is crucial to the general task of understanding. Building an explanation allows a system to determine which features of a situation or episode are causally relevant, and this determination is useful in learning generalizations.

Our approach to causal analysis and explanation differs from most other approaches, in particular the approach taken by most expert systems, in that we believe that explanation is a memory phenomenon. Wherever possible, an explaining system should retrieve an old explanation rather than build a new one. While building explanations by chaining together small pieces of causal knowledge increases flexibility, undirected backwards chaining causes a combinatoric explosion of connections to consider. Although not a problem in highly restricted domains, combinatoric explosion is rapidly aggravated by increasing knowledge. This violates our intuition that increasing knowledge and experience should facilitate building explanations, not make the process more difficult. On the other hand, if a system is based upon retrieving old explanations and applying them to new situations, a larger memory should make the system's job easier in that it is more likely to find an explanation that can be applied to the new situation with relatively little modification. We store old explanations in a template form, using a frame-like knowledge structure called an Explanation Pattern, or XP.

In the SWALE program, we have tested and refined the use of explanation in understanding. The key points of SWALE system include the following.

- Reuse Old Processing effort (retrieve versus build).
- Adapt knowledge to new situations.
- More flexible than fixed pattern instantiation.
- More efficient than building chains from scratch.
- Use Explanation Patterns (XPs): frozen explanatory inference chains.

These points are reflected in the algorithm used by SWALE to process new facts:

1. *Anomaly Detection:* Attempt to fit a new fact into memory. If successful DONE; otherwise an anomaly has been detected.
2. *XP Search:* Search for an XP that can be applied to explain the anomaly.
3. *XP Accepting:* Attempt to apply the retrieved explanations. If successful then skip to step 5.
4. *XP Tweaking:* If unable to apply XPs directly then attempt to tweak them into XPs that might apply better. If successful send these tweaked XPs back to step 3.
5. *XP Integration/Generalization:* If any results are accepted, integrate results back into memory, making appropriate generalizations.

Our work on explanation has been in two parts. We have completed a preliminary implementation of the algorithm above, and demonstrated it on a few examples. In addition, we are studying the types of pre-packaged explanations people actually use to resolve anomalous situations. We have gathered several hundred XPs, and are in the process of categorizing them. We anticipate that this work will soon result in a Yale Computer Science Department Technical Report.

Direct Memory Access Parsing

It is fairly standard now for serious language analyzers to make use of the notion of contextual expectation. That is, how we understand a sentence is influenced by what we expect to here. To do this, language analyzers need access to what memory is expecting. Unfortunately, most existing models of language analysis were developed before serious knowledge-based AI reasoning systems existed. As a result, parsers for intelligent databases or story understanding systems have poor or non-existent mechanisms for access to the knowledge contained in the underlying system. For example, in the slides depicting "Conceptual Analyzers" and "Memory-Based Analyzers" access to knowledge in memory is accomplished by passing copies of data structures back and forth. This restricts the kinds of information memory can provide the parser. It would be inefficient to pass many data structures containing high-frequency low-level inferences, such as "If a person picks up an object, that person is now holding that object," and a waste of effort to use them to pass low-probability high-level inferences, such as "If a person picks up an object, that person might have the goal of using that object for its normal purpose." Hence, standard parsers do their own low-level inferences, which requires duplicating the necessary knowledge. Standard parsers do without high-level inferences entirely.

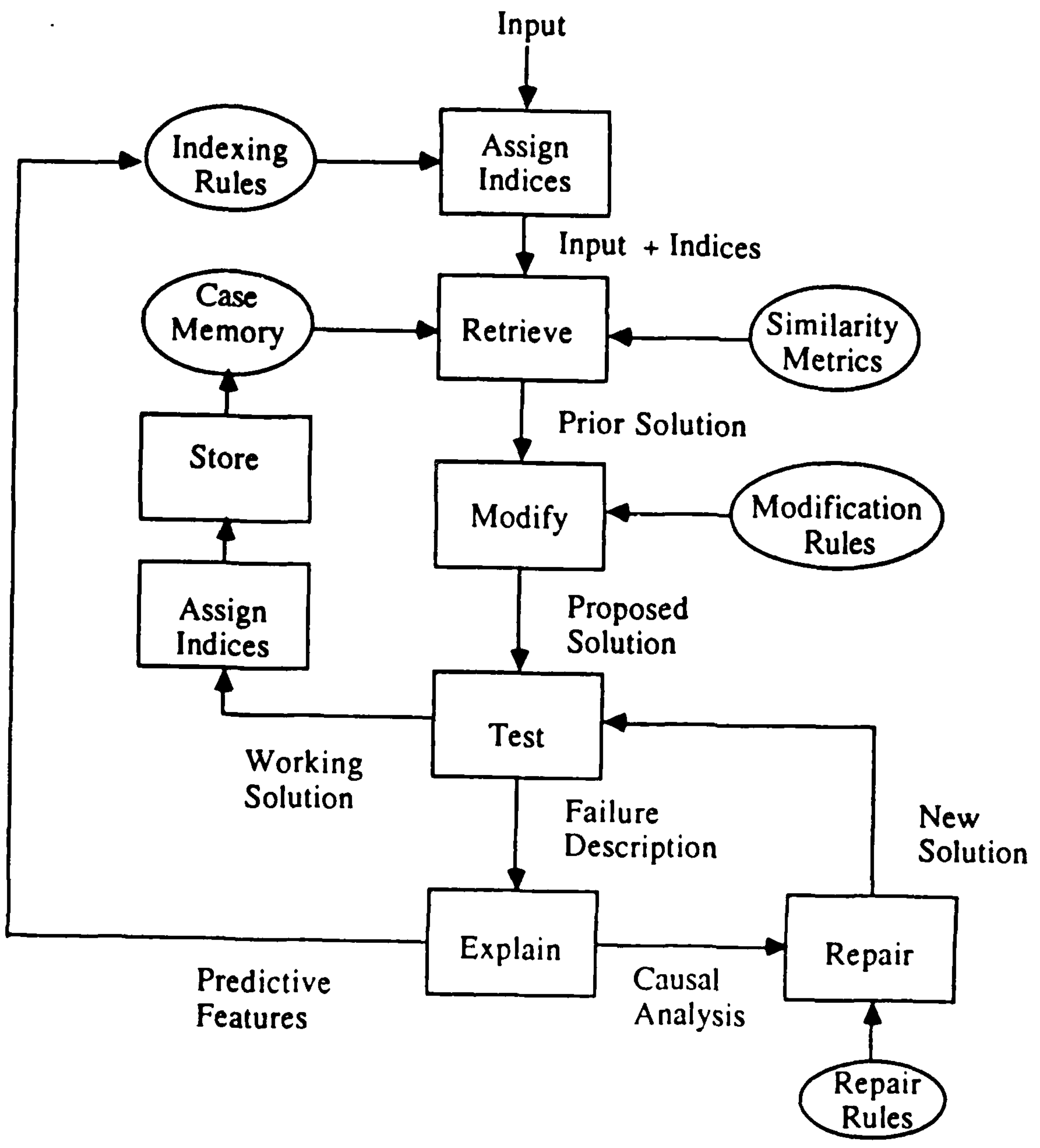
The Direct Memory Access Parser (DMAP) integrates the parsing process directly into standard memory search and inference processing. Again, the basic idea is simple: parsing is a memory search process that is guided by textual input. The goal of parsing is not to produce a meaning representation but to find relevant memory structures. This process is implemented with marker passing. Some memory structures are marked when predicted by inferences and others are marked when activated by text input. These markers are passed through the memory network and where they meet determines the relevant memory structures, i.e., the "parse" of the sentence. The promises of this approach are:

- Ambiguity is handled automatically by the marker intersection process.
- *Any* information available to the memory is available to the parser.
- It is impossible for the parser to produce ill-formed memory structures, since all the parser does is select existing memory structures.
- The representation of linguistic knowledge is simple enough to be reversible and used for language generation as well.

DMAP is, in some sense, a case-based approach to parsing. New texts are seen as variations on old texts, just as case-based reasoners see new problems as variations of old ones.

There are several implementations of DMAP. The oldest implementation has been in existence for 2 years and parses texts taken from newspapers and magazine articles discussing economics. The program not only reads and remembers economic arguments, but recognizes when arguments are variations of prior arguments, triggers inferences based on the variations found. DMAP is also being applied to a program that understands and asks questions about a story about terrorism, and to an intelligent self-organizing database of lung cancer slides.

Case-Based Reasoning



SWALE Algorithm

Anomaly Detection	(1) Attempt to fit story into memory. If successful DONE; otherwise an anomaly has been detected.
XP Search	(2) Search for an XP that can be applied to explain the anomaly
Accepting	(3) Attempt to apply XPs. If successful then skip to step 5
Tweaking	(4) If unable to apply XPs directly then attempt to tweak them into XPs that might apply better. If successful send these tweaked XPs back to step 3.
Integration/ Generalization	(5) If any results accepted, integrate results back into memory, making appropriate generalizations.

Some of SWALE's XPs

- **The Jim Fixx XP:**

Joggers jog a lot.
Jogging => exhaustion.
Exhaustion and heart defect => heart attack
A heart attack => death.

- **The Janis Joplin XP:**

Being a star performer => stress.
Being stressed-out => a need to escape.
Need to escape => taking recreational drugs.
Taking recreational drugs => overdose.
A drug overdose => death.

- **Useful Folk Knowledge**

Everybody knows that too much sex can kill you.

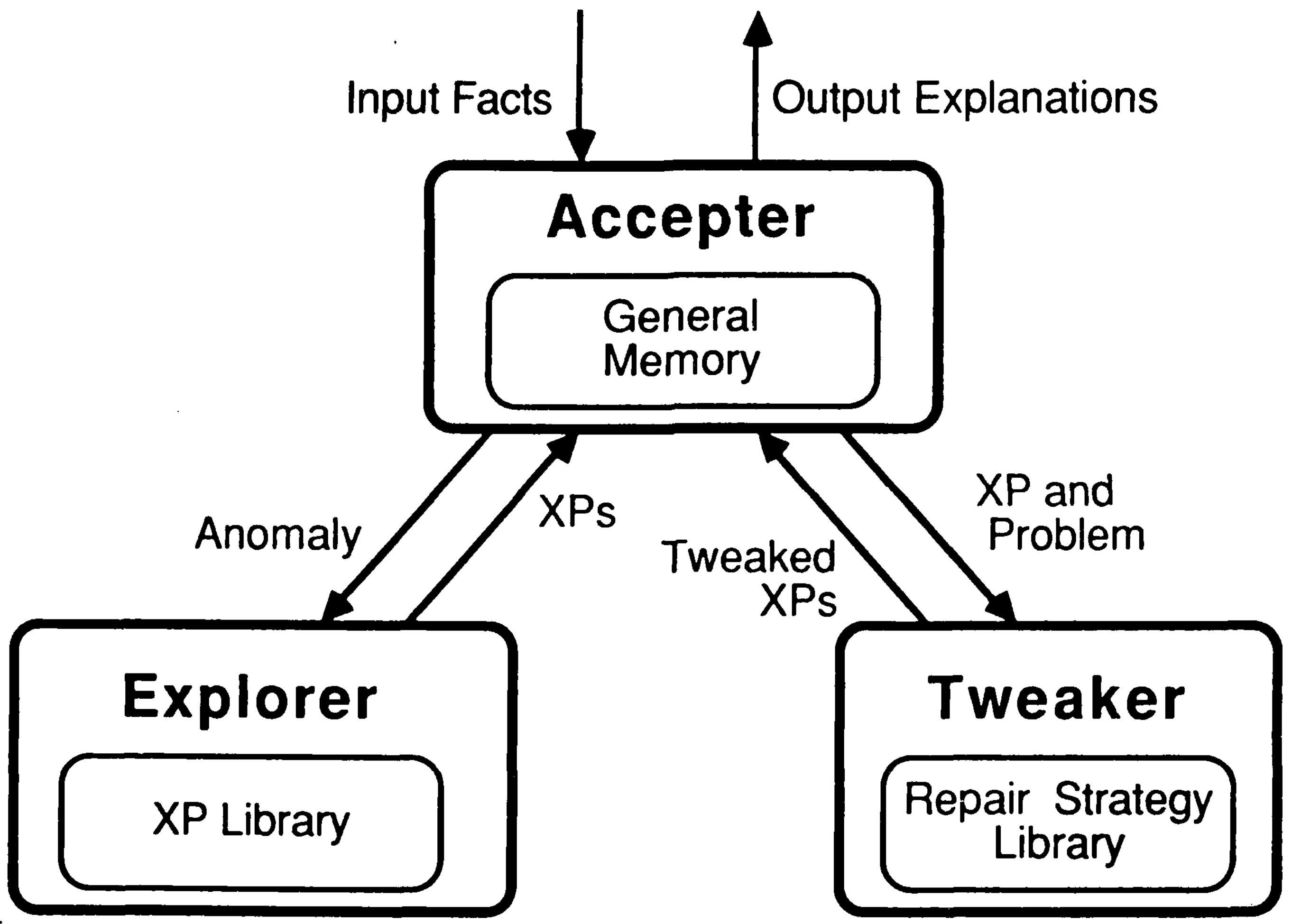
SWALE's Explanations:

Swale had a congenital heart defect.
The exertion of running in horse races strained his heart and brought out the latent defect.
He had a heart attack and died.

Swale's owner was giving him drugs to improve his performance.
He accidentally gave him an overdose.
The overdose killed him.

Swale was thinking about his future on the stud farm.
The excitement caused a heart attack.

SWALE Module Interconnection



Using Memory to Explain

- Reuse Old Processing Effort
- Adapt Knowledge to New Situations
- More Flexible than Pattern Instantiation
- More Efficient than Building Chains
- Use Explanation Patterns (XPs):

  Frozen Explanatory Inference Chains

An Input Story:

Swale was a successful three-year-old racehorse.

Swale won the Belmont Stakes.

Swale died a week later.

An Explanation Pattern

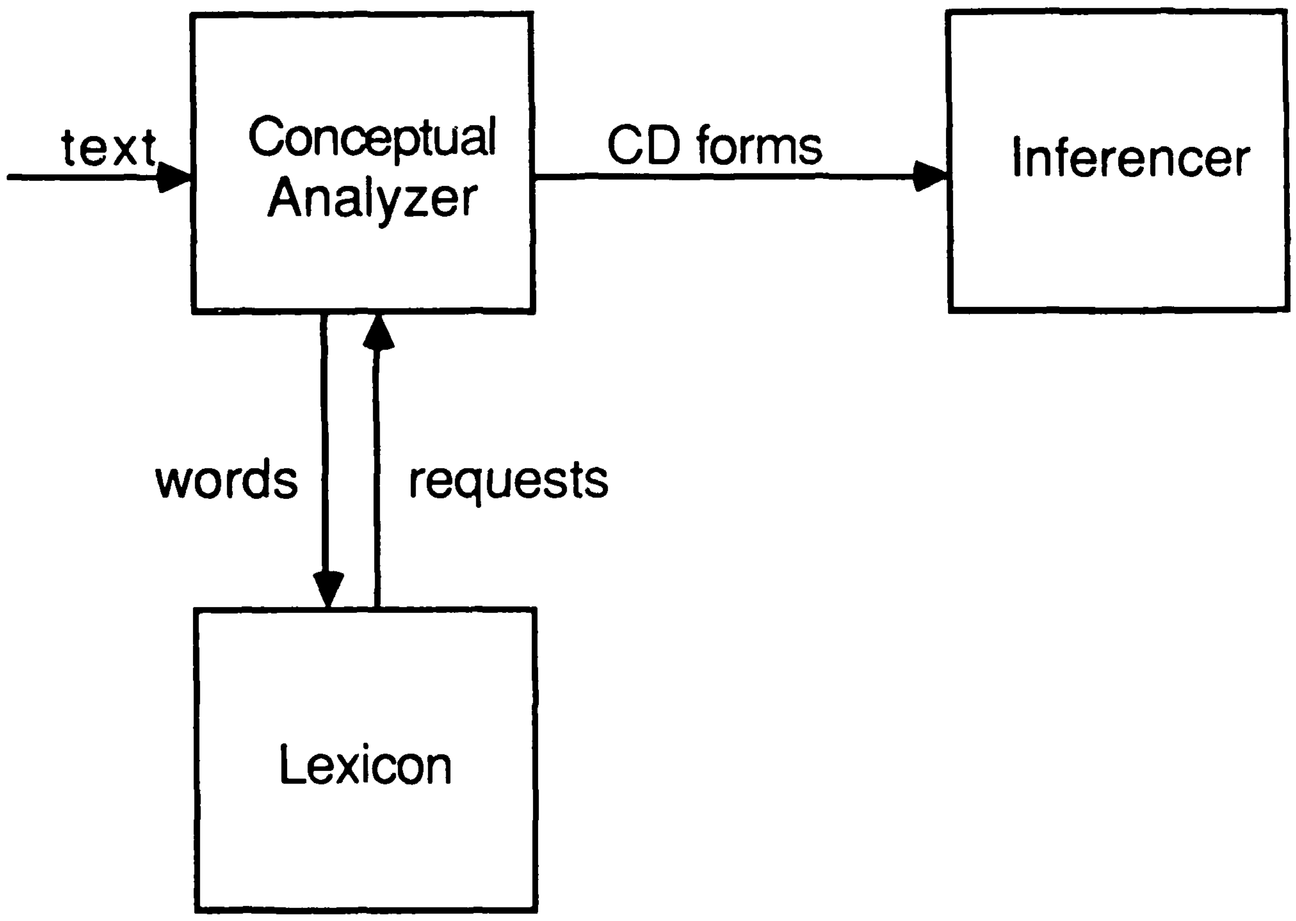
- Entry conditions : When is it applicable
- Relevance conditions: When is it relevant

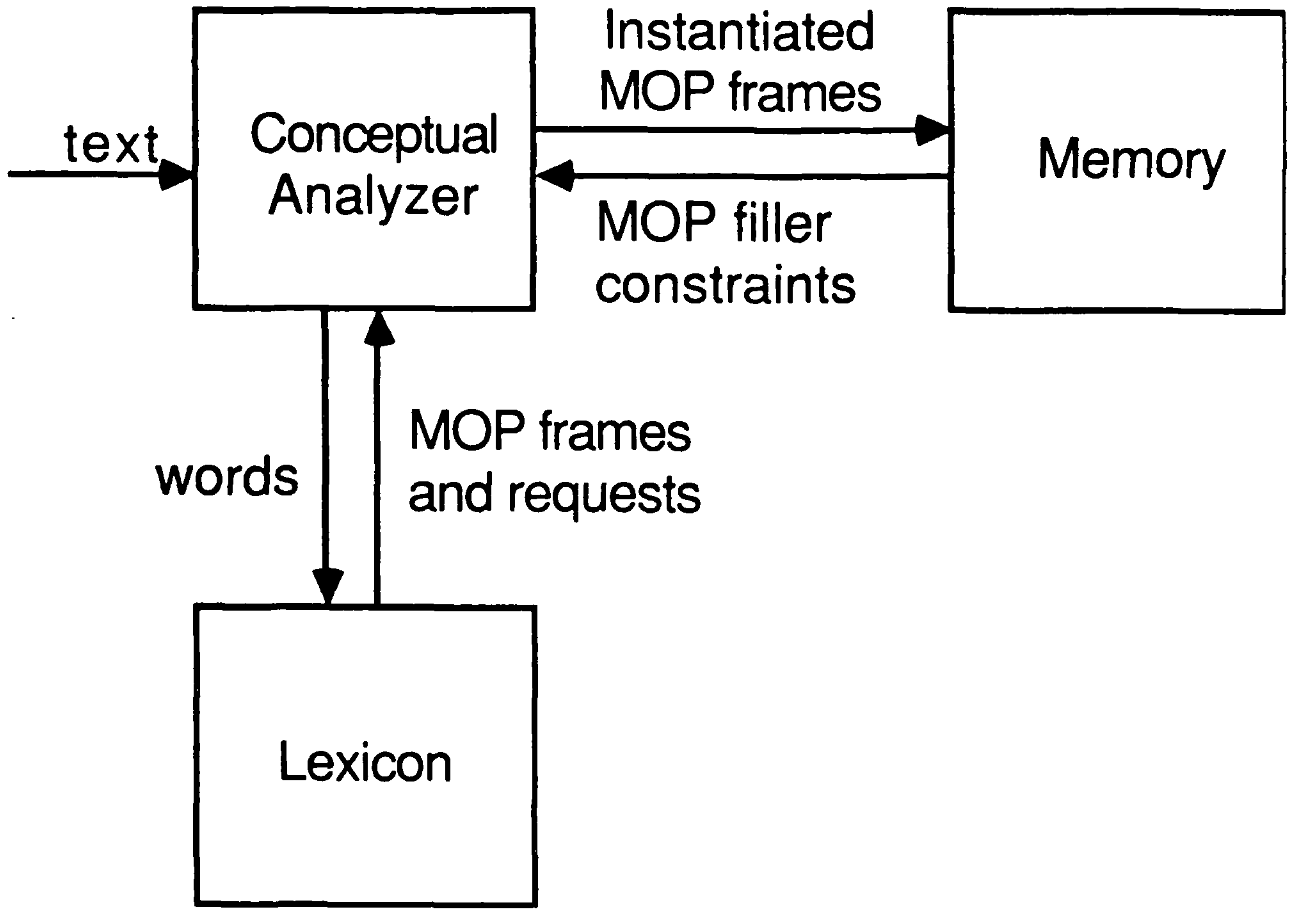
  or a candidate for modification.
- Causal Chain

DIRECT MEMORY ACCESS PARSING

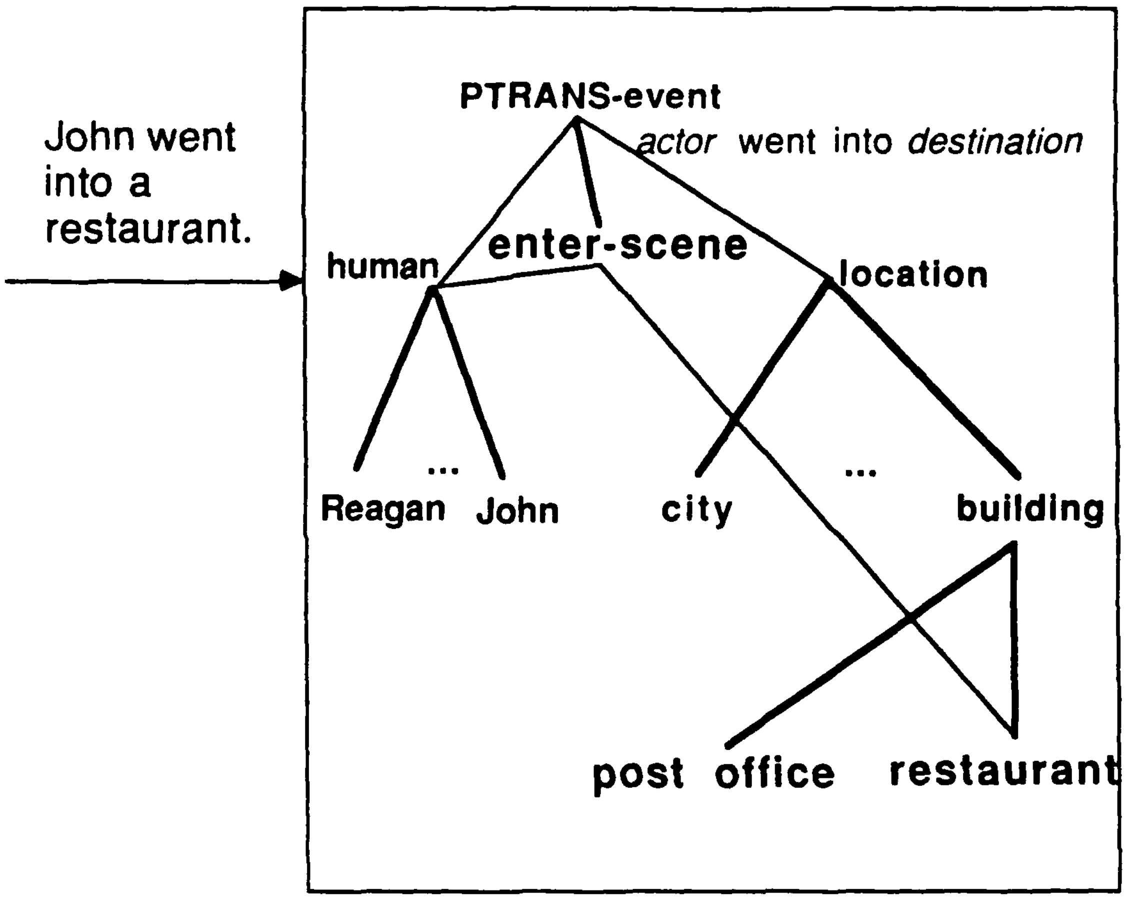
- only memory structure instantiation, no other form construction
- lexically-driven memory search process
- lexicon stored attached directly to concepts in memory

Parsing is treated as "recognize and record" rather than "build and store." A new text is either recognized as saying the same thing as an old text, or as being a specialization of a generalized text analysis. In the extreme, a whole article may be parsed as "same as article-25," if nothing new is said.


text
Conceptual Analyzer
CD forms
Inferencer
words
requests
Lexicon


text
Conceptual Analyzer
Instantiated MOP frames
Memory
MOP filler constraints
words
MOP frames and requests
Lexicon

MEMORY SEARCH PROCESSES



Direct Memory Access Parsing

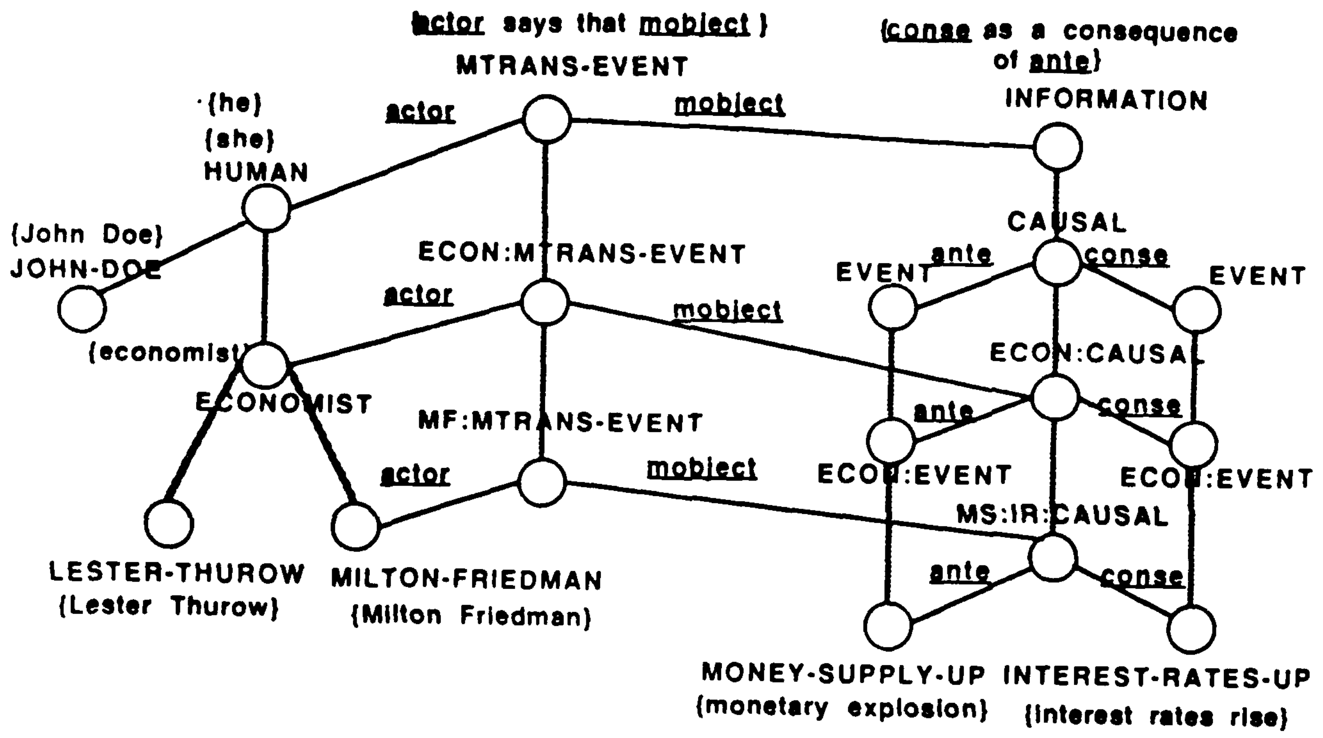
Memory Organization Packets (MOPs)

- Abstraction hierarchy (IS-A)
- Packaging hierarchy (PART-OF)

MOPs have zero or more concept sequences attached

- MTRANS-EVENT {actor says that mobject}
- MILTON-FRIEDMAN {Milton Friedman}

Concept lexicon consists simply of pointers from words, concepts, etc., to concept sequences.


{actor says that mobject}
MTRANS-EVENT
{conse as a consequence of ante}
INFORMATION
{he}
{she}
HUMAN
actor
mobject
{John Doe}
JOHN-DOE
ECON:MTRANS-EVENT
actor
mobject
CAUSAL
ante
conse
EVENT
EVENT
{economist}
ECONOMIST
ECON:CAUSAL
ante
conse
MF:MTRANS-EVENT
actor
mobject
ECON:EVENT
ECON:EVENT
MS:IR:CAUSAL
ante
conse
LESTER-THUROW
{Lester Thurow}
MILTON-FRIEDMAN
{Milton Friedman}
MONEY-SUPPLY-UP
{monetary explosion}
INTEREST-RATES-UP
{interest rates rise}

Markers

Two kinds of markers

- Activation markers
  - ◊ placed on words as they are read
  - ◊ placed on MOPs with fully activated concept sequences
  - ◊ passed up the abstraction hierarchy
- Prediction markers
  - ◊ placed on concept sequences indexed (in the concept lexicon) by activated structures
  - ◊ passed to concept sequence elements

Publications

Articles

Bain, W. A Case-based Reasoning System for Subjective Assessment. In Proceedings of the *Fifth National Conference on Artificial Intelligence*. Philadelphia, PA, August, 1986.

Kass, A. Modifying Explanations to Understand Stories. In Proceedings of the *Eighth Annual Conference of the Cognitive Science Society*. Amherst, MA, August, 1986.

Kass, A., Leake, D., and Owens, C. SWALE, A Program that Explains. In *Explanation Patterns: Understanding Mechanically and Creatively*, by R. C. Schank, In Press.

Martin, C., and Riesbeck, C. Uniform Parsing and Inferencing for Learning. In Proceedings of the *Fifth National Conference on Artificial Intelligence*. Philadelphia, PA, August, 1986.

Owens, C., and Leake, D. Organizing Memory for Explanation. In Proceedings of the *Eighth Annual Conference of the Cognitive Science Society*. Amherst, MA, August, 1986.

Riesbeck, C. Direct Memory Access Parsing. In proceedings of the *Second Annual Workshop on Theoretical Issues in Conceptual Information Processing*. New Haven, CT, May, 1985.

Riesbeck, C. Parsing, Expectation-Driven. In S. Shapiro (ed.) *Encyclopedia of Artificial Intelligence*. Forthcoming.

Riesbeck, C. From Conceptual Analyzer to Direct Memory Access Parsing: An Overview. In N. Sharkey (ed.) *Advances in Cognitive Science*. Forthcoming.

Riesbeck, C., and Martin, C. Direct Memory Access Parsing. In Kolodner and Riesbeck (eds.), *Experience, Memory, and Reasoning*. Lawrence Erlbaum Associates, Hillsdale, NJ. In press.

Riesbeck, C., and Martin, C. Towards Completely Integrated Parsing and Inferencing. In Proceedings of the *Eighth Annual Conference of the Cognitive Science Society*. Amherst, MA, August, 1986.

Schank, R., Collins, G. and Hunter, L. On the Failure of Inductive Category Formation as a Model of Learning. *Behavioral and Brain Sciences*, in press.

Schank, R., and Owens, C. Understanding by Explaining Expectation Failures. To appear in *Communication Failures In Discourse*, Edited by Ronan Reilly, North-Holland, In Press.

Schank, R., and Leake, D. Computer Understanding and Creativity. Invited paper for 1986 Conference of the International Federation for Information Processing. Dublin, Ireland.

Technical Reports

Schank, R. and Riesbeck, C. *Explanation: A Second Pass*. Technical Report 384. Yale Department of Computer Science. August, 1985.

Schank, R. and Riesbeck, C. *Questions and Thought*. Technical Report 385 Yale Department of Computer Science. August, 1985.

Books

Charniak, E., Riesbeck, C., McDermott, D., and Meehan, J. *Artificial Intelligence Programming. Second Edition.* Lawrence Erlbaum Associates, Hillsdale, NJ. In press.

Riesbeck, C. and Kolodner, J. (editors). *Experience, Memory, and Reasoning.* Lawrence Erlbaum Associates, Hillsdale, NJ. In press.

Schank, R. *Explanation Patterns: Understanding Mechanically and Creatively.* Lawrence Erlbaum Associates, Hillsdale, NJ. 1986.

Schank, R. and Riesbeck, C. *Inside Computer Learning*, Lawrence Erlbaum Associates, Hillsdale, NJ. In preparation.

Slade, S. *The T Programming Language: A Dialect of LISP.* Prentice-Hall, Englewood Cliffs, NJ. In press.

Ph.D. Dissertations

Bain, William. *Case-based Reasoning: A Computer Model of Subjective Assessment.*

Hammond, Kristian. *Case-based Planning: Viewing planning as a memory task*

Major Research Efforts

There are three major areas of accomplishments in recent AFOSR sponsored AI research at Yale.

Case-based reasoning

Theoretical Background:

The current standard model of expert reasoning is rule-based. Expertise is encoded in hundreds to independent *if–then* rules. While this technique is an improvement over previous ad hoc coding methods, it has several major inadequacies as an approach to serious expertise:

- It is hard for programmers to extend and debug rules because they are not organized into a larger coherent structure.
- It is hard for domain experts to translate their expertise into rule form.
- It is hard for the program itself to learn, i.e., to automatically generate new rules from its own problem-solving experiences.

At Yale, we have developed an alternative to rule-based reasoning which we call ***case-based reasoning***. We believe case-based reasoning more accurately models human expertise and better supports learning and knowledge acquisition. The basic idea is simple: new problems are solved by adapting solutions to previously solved problems. A case-based reasoner is initialized with a case library of basic problems and solutions, which become the foundation of future problem-solving. The program adapts cases to new problems and adds successful adaptations to the library.

Such an approach has several promising advantages:

- Problem-solving behavior improves automatically, as solutions to common problems are saved for future use.
- Using simple substitution rules, the program can adapt and use a library solution that would be very difficult, if not impossible, to generate from scratch with a rule-based approach.
- Experts can extend the program simply by adding new example solutions to the library.

While case-based reasoning is simple in concept, there are of course many subtle and difficult design issues that have to be resolved to make it work. One major achievement in the past two years at Yale has been the development of an explicit structure for the case-based reasoning process, as shown in the slide containing the flowchart entitled "Case-Based Reasoning." This process description summarizes results from the JUDGE (Bain 1986) AFOSR project as well as from other projects such as CHEF (Hammond 1986) and COACH (Collins forthcoming). The flow of control in the process (represented by the arrows and boxes) is task-independent, while the particular knowledge bases (represented by the ovals) are task-specific.

There are two kinds of learning currently implemented in our programs, as indicated by arrows leading *into* knowledge bases. First, new solutions are stored in case memory for retrieval by future problem descriptions. This is "learning from success." Second, new rules for assigning indices to problem descriptions are stored whenever a retrieved solution fails to solve a problem. Such failures indicate that some feature in the problem description was overlooked. By checking for this feature in the future, inappropriate cases will not be retrieved. This is "learning from failure." Both kinds of learning are crucial to case-based reasoning.

This model is implemented in JUDGE, as well as several other problem-solvers. JUDGE deals with problems of subjective assessment. Subjective assessment is a kind of task that people have to do all the time, including appraisals of worth (houses, art, etc.,) judgments of merit (in gymnastics, the office, etc.,) and evaluations of goals (in self and others). In all subjective assessment problems, the explicitly known rules for assessment are very incomplete, and there is scant feedback regarding the accuracy of the assessment.

Case-based reasoning fits in very well here because it does not require a completely specified rule set. Instead it can use a small library of reasonable assessment solutions. In place of "correct/incorrect" feedback, JUDGE's adaptation rules work to maintain consistency with past assessments.

The JUDGE program models the subjective assessment task of judicial sentencing. That is, given a description of an event, such as a fight between two people that ended in a death or serious injury, where someone has been convicted of a criminal act, JUDGE determines a sentence that meets the rules of the law and is consistent with prior sentencing behavior. Each sentencing event is added to the case memory and used, when relevant, in later sentencing.

Explanation-based learning

An important extension of case-based reasoning is learning through explanation. We use the term explanation to refer to the construction of a causal model of an event or episode rather than to any particular communication a system might have with its users. Internal explanations are important so that a system can reason on the basis of the causal relationships it has discovered. Examples of tasks requiring this kind of explanation include adaptive recovery from failures in robot planning and navigation, diagnosis of faults in various hardware and software systems, and flexible behavior in the face of unexpected situations.

Furthermore, we believe that explanation is crucial to the general task of understanding. Building an explanation allows a system to determine which features of a situation or episode are causally relevant, and this determination is useful in learning generalizations.

Our approach to causal analysis and explanation differs from most other approaches, in particular the approach taken by most expert systems, in that we believe that explanation is a memory phenomenon. Wherever possible, an explaining system should retrieve an old explanation rather than build a new one. While building explanations by chaining together small pieces of causal knowledge increases flexibility, undirected backwards chaining causes a combinatoric explosion of connections to consider. Although not a problem in highly restricted domains, combinatoric explosion is rapidly aggravated by increasing knowledge. This violates our intuition that increasing knowledge and experience should facilitate building explanations, not make the process more difficult. On the other hand, if a system is based upon retrieving old explanations and applying them to new situations, a larger memory should make the system's job easier in that it is more likely to find an explanation that can be applied to the new situation with relatively little modification. We store old explanations in a template form, using a frame-like knowledge structure called an Explanation Pattern, or XP.

In the SWALE program, we have tested and refined the use of explanation in understanding. The key points of SWALE system include the following.

- Reuse Old Processing effort (retrieve versus build).
- Adapt knowledge to new situations.
- More flexible than fixed pattern instantiation.
- More efficient than building chains from scratch.
- Use Explanation Patterns (XPs): frozen explanatory inference chains.

These points are reflected in the algorithm used by SWALE to process new facts:

1. *Anomaly Detection:* Attempt to fit a new fact into memory. If successful DONE: otherwise an anomaly has been detected.
2. *XP Search:* Search for an XP that can be applied to explain the anomaly.
3. *XP Accepting:* Attempt to apply the retrieved explanations. If successful then skip to step 5.
4. *XP Tweaking:* If unable to apply XPs directly then attempt to tweak them into XPs that might apply better. If successful send these tweaked XPs back to step 3.
5. *XP Integration/Generalization:* If any results are accepted, integrate results back into memory, making appropriate generalizations.

Our work on explanation has been in two parts. We have completed a preliminary implementation of the algorithm above, and demonstrated it on a few examples. In addition, we are studying the types of pre-packaged explanations people actually use to resolve anomalous situations. We have gathered several hundred XPs, and are in the process of categorizing them. We anticipate that this work will soon result in a Yale Computer Science Department Technical Report.

Direct Memory Access Parsing

It is fairly standard now for serious language analyzers to make use of the notion of contextual expectation. That is, how we understand a sentence is influenced by what we expect to here. To do this, language analyzers need access to what memory is expecting. Unfortunately, most existing models of language analysis were developed before serious knowledge-based AI reasoning systems existed. As a result, parsers for intelligent databases or story understanding systems have poor or non-existent mechanisms for access to the knowledge contained in the underlying system. For example, in the slides depicting "Conceptual Analyzers" and "Memory-Based Analyzers" access to knowledge in memory is accomplished by passing copies of data structures back and forth. This restricts the kinds of information memory can provide the parser. It would be inefficient to pass many data structures containing high-frequency low-level inferences, such as "If a person picks up an object, that person is now holding that object," and a waste of effort to use them to pass low-probability high-level inferences, such as "If a person picks up an object, that person might have the goal of using that object for its normal purpose." Hence, standard parsers do their own low-level inferences, which requires duplicating the necessary knowledge. Standard parsers do without high-level inferences entirely.

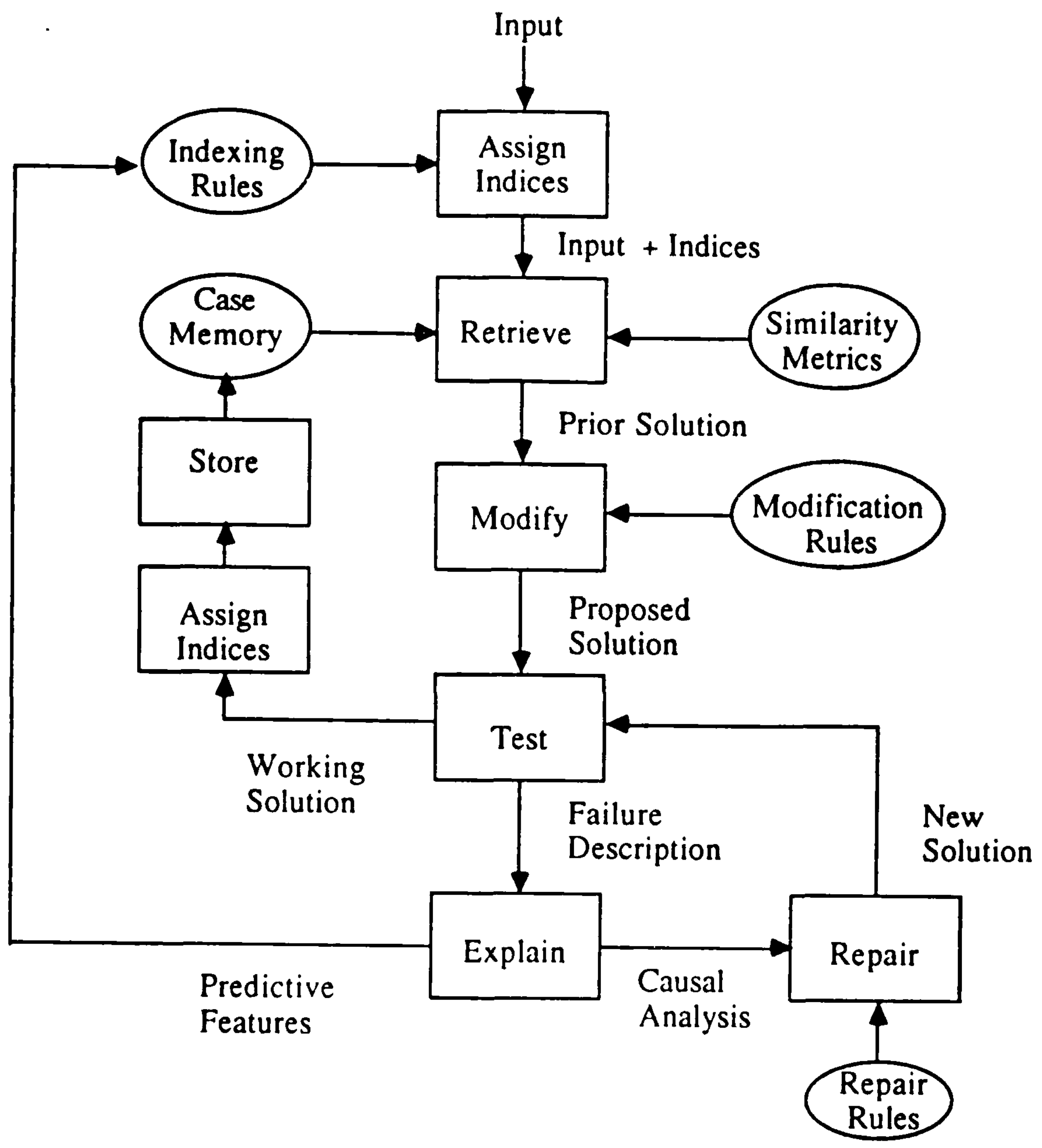
The Direct Memory Access Parser (DMAP) integrates the parsing process directly into standard memory search and inference processing. Again, the basic idea is simple: parsing is a memory search process that is guided by textual input. The goal of parsing is not to produce a meaning representation but to find relevant memory structures. This process is implemented with marker passing. Some memory structures are marked when predicted by inferences and others are marked when activated by text input. These markers are passed through the memory network and where they meet determines the relevant memory structures, i.e., the "parse" of the sentence. The promises of this approach are:

- Ambiguity is handled automatically by the marker intersection process.
- *Any* information available to the memory is available to the parser.
- It is impossible for the parser to produce ill-formed memory structures, since all the parser does is select existing memory structures.
- The representation of linguistic knowledge is simple enough to be reversible and used for language generation as well.

DMAP is, in some sense, a case-based approach to parsing. New texts are seen as variations on old texts, just as case-based reasoners see new problems as variations of old ones.

There are several implementations of DMAP. The oldest implementation has been in existence for 2 years and parses texts taken from newspapers and magazine articles discussing economics. The program not only reads and remembers economic arguments, but recognizes when arguments are variations of prior arguments, triggers inferences based on the variations found. DMAP is also being applied to a program that understands and asks questions about a story about terrorism, and to an intelligent self-organizing database of lung cancer slides.

Case-Based Reasoning



SWALE Algorithm

Anomaly Detection	(1) Attempt to fit story into memory. If successful DONE; otherwise an anomaly has been detected.
XP Search	(2) Search for an XP that can be applied to explain the anomaly
Accepting	(3) Attempt to apply XPs. If successful then skip to step 5
Tweaking	(4) If unable to apply XPs directly then attempt to tweak them into XPs that might apply better. If successful send these tweaked XPs back to step 3.
Integration/ Generalization	(5) If any results accepted, integrate results back into memory, making appropriate generalizations.

Some of SWALE's XPs

- **The Jim Fixx XP:**

Joggers jog a lot.
Jogging => exhaustion.
Exhaustion and heart defect => heart attack
A heart attack => death.

- **The Janis Joplin XP:**

Being a star performer => stress.
Being stressed-out => a need to escape.
Need to escape => taking recreational drugs.
Taking recreational drugs => overdose.
A drug overdose => death.

- **Useful Folk Knowledge**

Everybody knows that too much sex can kill you.

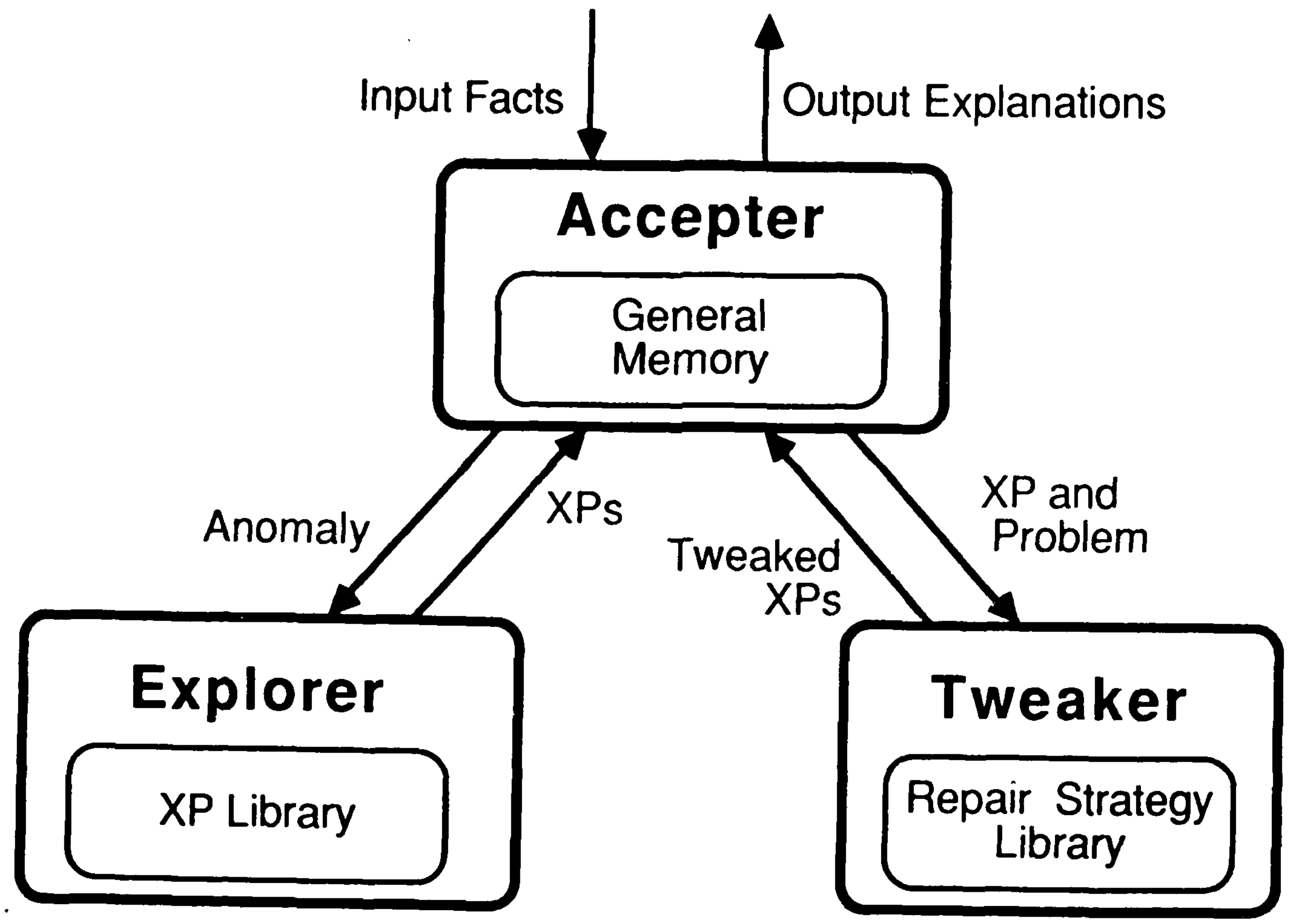
SWALE's Explanations:

Swale had a congenital heart defect.
The exertion of running in horse races strained his heart and brought out the latent defect.
He had a heart attack and died.

Swale's owner was giving him drugs to improve his performance.
He accidentally gave him an overdose.
The overdose killed him.

Swale was thinking about his future on the stud farm.
The excitement caused a heart attack.

SWALE Module Interconnection



Using Memory to Explain

- Reuse Old Processing Effort
- Adapt Knowledge to New Situations
- More Flexible than Pattern Instantiation
- More Efficient than Building Chains
- Use Explanation Patterns (XPs):

  Frozen Explanatory Inference Chains

An Input Story:

Swale was a successful three-year-old racehorse.

Swale won the Belmont Stakes.

Swale died a week later.

An Explanation Pattern

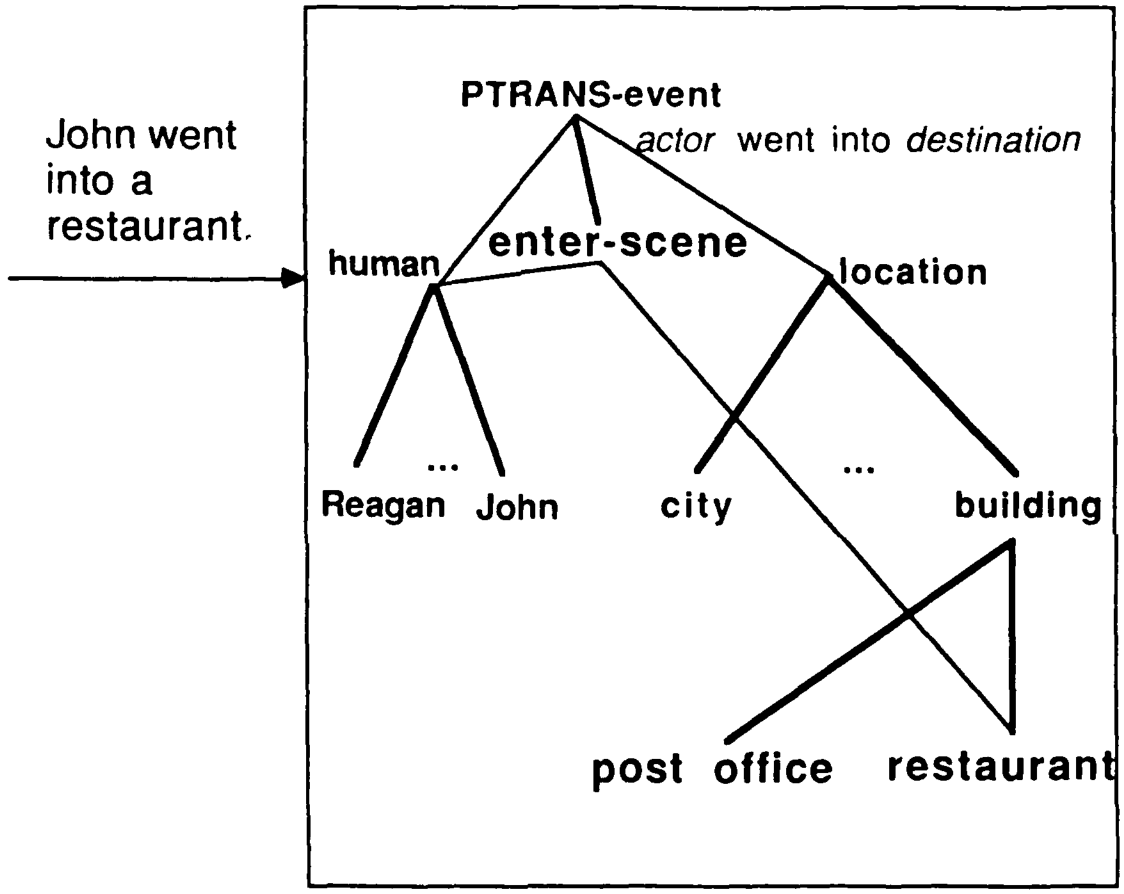
- Entry conditions : When is it applicable
- Relevance conditions: When is it relevant or a candidate for modification.
- Causal Chain

DIRECT MEMORY ACCESS PARSING

- only memory structure instantiation, no other form construction
- lexically-driven memory search process
- lexicon stored attached directly to concepts in memory

Parsing is treated as "recognize and record" rather than "build and store." A new text is either recognized as saying the same thing as an old text, or as being a specialization of a generalized text analysis. In the extreme, a whole article may be parsed as "same as article-25," if nothing new is said.

MEMORY SEARCH PROCESSES



Direct Memory Access Parsing

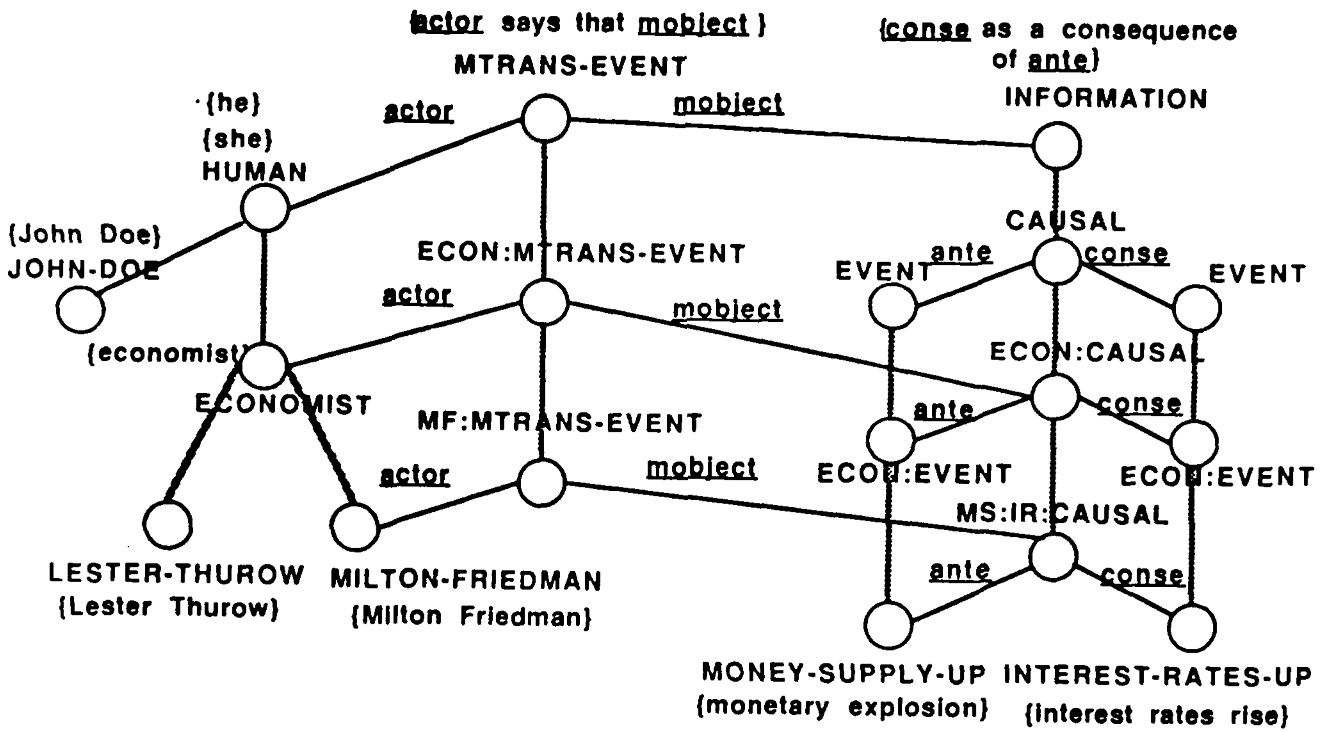
Memory Organization Packets (MOPs)

- Abstraction hierarchy (IS-A)
- Packaging hierarchy (PART-OF)

MOPs have zero or more concept sequences attached

- MTRANS-EVENT {actor says that mobject}
- MILTON-FRIEDMAN {Milton Friedman}

Concept lexicon consists simply of pointers from words, concepts, etc., to concept sequences.


{actor says that mobject}
MTRANS-EVENT
{conse as a consequence of ante}
INFORMATION
{he}
{she}
HUMAN
actor
mobject
{John Doe}
JOHN-DOE
CAUSAL
ante
conse
EVENT
EVENT
ECON:MTRANS-EVENT
actor
mobject
{economist}
ECONOMIST
ECON:CAUSAL
ante
conse
MF:MTRANS-EVENT
actor
mobject
ECON:EVENT
ECON:EVENT
MS:IR:CAUSAL
ante
conse
LESTER-THUROW
{Lester Thurow}
MILTON-FRIEDMAN
{Milton Friedman}
MONEY-SUPPLY-UP
{monetary explosion}
INTEREST-RATES-UP
{interest rates rise}

Markers

Two kinds of markers

- Activation markers
  - ◊ placed on words as they are read
  - ◊ placed on MOPs with fully activated concept sequences
  - ◊ passed up the abstraction hierarchy
- Prediction markers
  - ◊ placed on concept sequences indexed (in the concept lexicon) by activated structures
  - ◊ passed to concept sequence elements

Publications

Articles

Bain, W. A Case-based Reasoning System for Subjective Assessment. In Proceedings of the *Fifth National Conference on Artificial Intelligence.* Philadelphia, PA, August, 1986.

Kass, A. Modifying Explanations to Understand Stories. In Proceedings of the *Eighth Annual Conference of the Cognitive Science Society.* Amherst, MA, August, 1986.

Kass, A., Leake, D., and Owens, C. SWALE, A Program that Explains. In *Explanation Patterns: Understanding Mechanically and Creatively,* by R. C. Schank, In Press.

Martin, C., and Riesbeck, C. Uniform Parsing and Inferencing for Learning. In Proceedings of the *Fifth National Conference on Artificial Intelligence.* Philadelphia, PA, August, 1986.

Owens, C., and Leake, D. Organizing Memory for Explanation. In Proceedings of the *Eighth Annual Conference of the Cognitive Science Society.* Amherst, MA, August, 1986.

Riesbeck, C. Direct Memory Access Parsing. In proceedings of the *Second Annual Workshop on Theoretical Issues in Conceptual Information Processing.* New Haven, CT, May, 1985.

Riesbeck, C. Parsing, Expectation-Driven. In S. Shapiro (ed.) *Encyclopedia of Artificial Intelligence.* Forthcoming.

Riesbeck, C. From Conceptual Analyzer to Direct Memory Access Parsing: An Overview. In N. Sharkey (ed.) *Advances in Cognitive Science.* Forthcoming.

Riesbeck, C., and Martin, C. Direct Memory Access Parsing. In Kolodner and Riesbeck (eds.), *Experience, Memory, and Reasoning.* Lawrence Erlbaum Associates, Hillsdale, NJ. In press.

Riesbeck, C., and Martin, C. Towards Completely Integrated Parsing and Inferencing. In Proceedings of the *Eighth Annual Conference of the Cognitive Science Society.* Amherst, MA, August, 1986.

Schank, R., Collins, G. and Hunter, L. On the Failure of Inductive Category Formation as a Model of Learning. *Behavioral and Brain Sciences,* in press.

Schank, R., and Owens, C. Understanding by Explaining Expectation Failures. To appear in *Communication Failures In Discourse,* Edited by Ronan Reilly, North-Holland, In Press.

Schank, R., and Leake, D. Computer Understanding and Creativity. Invited paper for 1986 Conference of the International Federation for Information Processing. Dublin, Ireland.

Technical Reports

Schank, R. and Riesbeck, C. *Explanation: A Second Pass.* Technical Report 384, Yale Department of Computer Science. August, 1985.

Schank, R. and Riesbeck, C. *Questions and Thought.* Technical Report 385 Yale Department of Computer Science. August, 1985.

Books

Charniak, E., Riesbeck, C., McDermott, D., and Meehan, J. *Artificial Intelligence Programming. Second Edition.* Lawrence Erlbaum Associates, Hillsdale, NJ. In press.

Riesbeck, C. and Kolodner, J. (editors). *Experience, Memory, and Reasoning.* Lawrence Erlbaum Associates, Hillsdale, NJ. In press.

Schank, R. *Explanation Patterns: Understanding Mechanically and Creatively.* Lawrence Erlbaum Associates, Hillsdale, NJ. 1986.

Schank, R. and Riesbeck, C. *Inside Computer Learning*, Lawrence Erlbaum Associates, Hillsdale, NJ. In preparation.

Slade, S. *The T Programming Language: A Dialect of LISP.* Prentice-Hall, Englewood Cliffs, NJ. In press.

Ph.D. Dissertations

Bain, William. *Case-based Reasoning: A Computer Model of Subjective Assessment.*

Hammond, Kristian. *Case-based Planning: Viewing planning as a memory task*

Major Research Efforts

There are three major areas of accomplishments in recent AFOSR sponsored AI research at Yale.

Case-based reasoning

Theoretical Background:

The current standard model of expert reasoning is rule-based. Expertise is encoded in hundreds to independent *if-then* rules. While this technique is an improvement over previous ad hoc coding methods, it has several major inadequacies as an approach to serious expertise:

- It is hard for programmers to extend and debug rules because they are not organized into a larger coherent structure.
- It is hard for domain experts to translate their expertise into rule form.
- It is hard for the program itself to learn, i.e., to automatically generate new rules from its own problem-solving experiences.

At Yale, we have developed an alternative to rule-based reasoning which we call ***case-based reasoning***. We believe case-based reasoning more accurately models human expertise and better supports learning and knowledge acquisition. The basic idea is simple: new problems are solved by adapting solutions to previously solved problems. A case-based reasoner is initialized with a case library of basic problems and solutions, which become the foundation of future problem-solving. The program adapts cases to new problems and adds successful adaptations to the library.

Such an approach has several promising advantages:

- Problem-solving behavior improves automatically, as solutions to common problems are saved for future use.
- Using simple substitution rules, the program can adapt and use a library solution that would be very difficult, if not impossible, to generate from scratch with a rule-based approach.
- Experts can extend the program simply by adding new example solutions to the library.

While case-based reasoning is simple in concept, there are of course many subtle and difficult design issues that have to be resolved to make it work. One major achievement in the past two years at Yale has been the development of an explicit structure for the case-based reasoning process, as shown in the slide containing the flowchart entitled "Case-Based Reasoning." This process description summarizes results from the JUDGE (Bain 1986) AFOSR project as well as from other projects such as CHEF (Hammond 1986) and COACH (Collins forthcoming). The flow of control in the process (represented by the arrows and boxes) is task-independent, while the particular knowledge bases (represented by the ovals) are task-specific.

There are two kinds of learning currently implemented in our programs, as indicated by arrows leading *into* knowledge bases. First, new solutions are stored in case memory for retrieval by future problem descriptions. This is "learning from success." Second, new rules for assigning indices to problem descriptions are stored whenever a retrieved solution fails to solve a problem. Such failures indicate that some feature in the problem description was overlooked. By checking for this feature in the future, inappropriate cases will not be retrieved. This is "learning from failure." Both kinds of learning are crucial to case-based reasoning.

This model is implemented in JUDGE, as well as several other problem-solvers. JUDGE deals with problems of subjective assessment. Subjective assessment is a kind of task that people have to do all the time, including appraisals of worth (houses, art, etc.,) judgments of merit (in gymnastics, the office, etc.,) and evaluations of goals (in self and others). In all subjective assessment problems, the explicitly known rules for assessment are very incomplete, and there is scant feedback regarding the accuracy of the assessment.

Case-based reasoning fits in very well here because it does not require a completely specified rule set. Instead it can use a small library of reasonable assessment solutions. In place of "correct/incorrect" feedback, JUDGE's adaptation rules work to maintain consistency with past assessments.

The JUDGE program models the subjective assessment task of judicial sentencing. That is, given a description of an event, such as a fight between two people that ended in a death or serious injury, where someone has been convicted of a criminal act, JUDGE determines a sentence that meets the rules of the law and is consistent with prior sentencing behavior. Each sentencing event is added to the case memory and used, when relevant, in later sentencing.

Explanation-based learning

An important extension of case-based reasoning is learning through explanation. We use the term explanation to refer to the construction of a causal model of an event or episode rather than to any particular communication a system might have with its users. Internal explanations are important so that a system can reason on the basis of the causal relationships it has discovered. Examples of tasks requiring this kind of explanation include adaptive recovery from failures in robot planning and navigation, diagnosis of faults in various hardware and software systems, and flexible behavior in the face of unexpected situations.

Furthermore, we believe that explanation is crucial to the general task of understanding. Building an explanation allows a system to determine which features of a situation or episode are causally relevant, and this determination is useful in learning generalizations.

Our approach to causal analysis and explanation differs from most other approaches, in particular the approach taken by most expert systems, in that we believe that explanation is a memory phenomenon. Wherever possible, an explaining system should retrieve an old explanation rather than build a new one. While building explanations by chaining together small pieces of causal knowledge increases flexibility, undirected backwards chaining causes a combinatoric explosion of connections to consider. Although not a problem in highly restricted domains, combinatoric explosion is rapidly aggravated by increasing knowledge. This violates our intuition that increasing knowledge and experience should facilitate building explanations, not make the process more difficult. On the other hand, if a system is based upon retrieving old explanations and applying them to new situations, a larger memory should make the system's job easier in that it is more likely to find an explanation that can be applied to the new situation with relatively little modification. We store old explanations in a template form, using a frame-like knowledge structure called an Explanation Pattern, or XP.

In the SWALE program, we have tested and refined the use of explanation in understanding. The key points of SWALE system include the following.

- Reuse Old Processing effort (retrieve versus build).
- Adapt knowledge to new situations.
- More flexible than fixed pattern instantiation.
- More efficient than building chains from scratch.
- Use Explanation Patterns (XPs): frozen explanatory inference chains.

These points are reflected in the algorithm used by SWALE to process new facts:

1. *Anomaly Detection:* Attempt to fit a new fact into memory. If successful DONE; otherwise an anomaly has been detected.
2. *XP Search:* Search for an XP that can be applied to explain the anomaly.
3. *XP Accepting:* Attempt to apply the retrieved explanations. If successful then skip to step 5.
4. *XP Tweaking:* If unable to apply XPs directly then attempt to tweak them into XPs that might apply better. If successful send these tweaked XPs back to step 3.
5. *XP Integration/Generalization:* If any results are accepted, integrate results back into memory, making appropriate generalizations.

Our work on explanation has been in two parts. We have completed a preliminary implementation of the algorithm above, and demonstrated it on a few examples. In addition, we are studying the types of pre-packaged explanations people actually use to resolve anomalous situations. We have gathered several hundred XPs, and are in the process of categorizing them. We anticipate that this work will soon result in a Yale Computer Science Department Technical Report.

Direct Memory Access Parsing

It is fairly standard now for serious language analyzers to make use of the notion of contextual expectation. That is, how we understand a sentence is influenced by what we expect to here. To do this, language analyzers need access to what memory is expecting. Unfortunately, most existing models of language analysis were developed before serious knowledge-based AI reasoning systems existed. As a result, parsers for intelligent databases or story understanding systems have poor or non-existent mechanisms for access to the knowledge contained in the underlying system. For example, in the slides depicting "Conceptual Analyzers" and "Memory-Based Analyzers" access to knowledge in memory is accomplished by passing copies of data structures back and forth. This restricts the kinds of information memory can provide the parser. It would be inefficient to pass many data structures containing high-frequency low-level inferences, such as "If a person picks up an object, that person is now holding that object," an l a waste of effort to use them to pass low-probability high-level inferences, such as "If a person picks up an object, that person might have the goal of using that object for its normal purpose." Hence, standard parsers do their own low-level inferences, which requires duplicating the necessary knowledge. Standard parsers do without high-level inferences entirely.

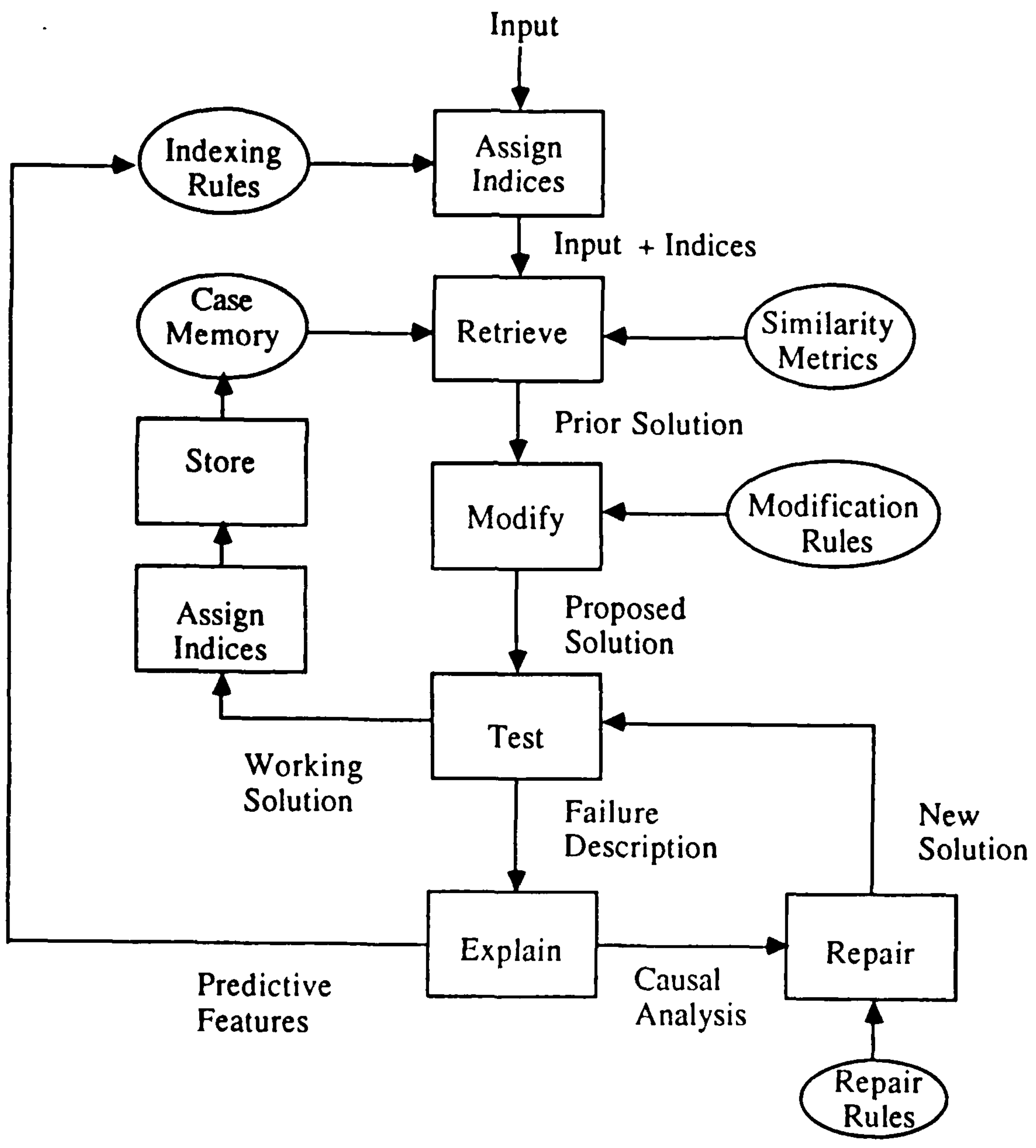
The Direct Memory Access Parser (DMAP) integrates the parsing process directly into standard memory search and inference processing. Again, the basic idea is simple: parsing is a memory search process that is guided by textual input. The goal of parsing is not to produce a meaning representation but to find relevant memory structures. This process is implemented with marker passing. Some memory structures are marked when predicted by inferences and others are marked when activated by text input. These markers are passed through the memory network and where they meet determines the relevant memory structures, i.e., the "parse" of the sentence. The promises of this approach are:

- Ambiguity is handled automatically by the marker intersection process.
- *Any* information available to the memory is available to the parser.
- It is impossible for the parser to produce ill-formed memory structures, since all the parser does is select existing memory structures.
- The representation of linguistic knowledge is simple enough to be reversible and used for language generation as well.

DMAP is, in some sense, a case-based approach to parsing. New texts are seen as variations on old texts, just as case-based reasoners see new problems as variations of old ones.

There are several implementations of DMAP. The oldest implementation has been in existence for 2 years and parses texts taken from newspapers and magazine articles discussing economics. The program not only reads and remembers economic arguments, but recognizes when arguments are variations of prior arguments, triggers inferences based on the variations found. DMAP is also being applied to a program that understands and asks questions about a story about terrorism, and to an intelligent self-organizing database of lung cancer slides.

Case-Based Reasoning



SWALE Algorithm

Anomaly Detection	(1) Attempt to fit story into memory. If successful DONE; otherwise an anomaly has been detected.
XP Search	(2) Search for an XP that can be applied to explain the anomaly
Accepting	(3) Attempt to apply XPs. If successful then skip to step 5
Tweaking	(4) If unable to apply XPs directly then attempt to tweak them into XPs that might apply better. If successful send these tweaked XPs back to step 3.
Integration/ Generalization	(5) If any results accepted, integrate results back into memory, making appropriate generalizations.

Some of SWALE's XPs

- **The Jim Fixx XP:**

Joggers jog a lot.
Jogging => exhaustion.
Exhaustion and heart defect => heart attack
A heart attack => death.

- **The Janis Joplin XP:**

Being a star performer => stress.
Being stressed-out => a need to escape.
Need to escape => taking recreational drugs.
Taking recreational drugs => overdose.
A drug overdose => death.

- **Useful Folk Knowledge**

Everybody knows that too much sex can kill you.

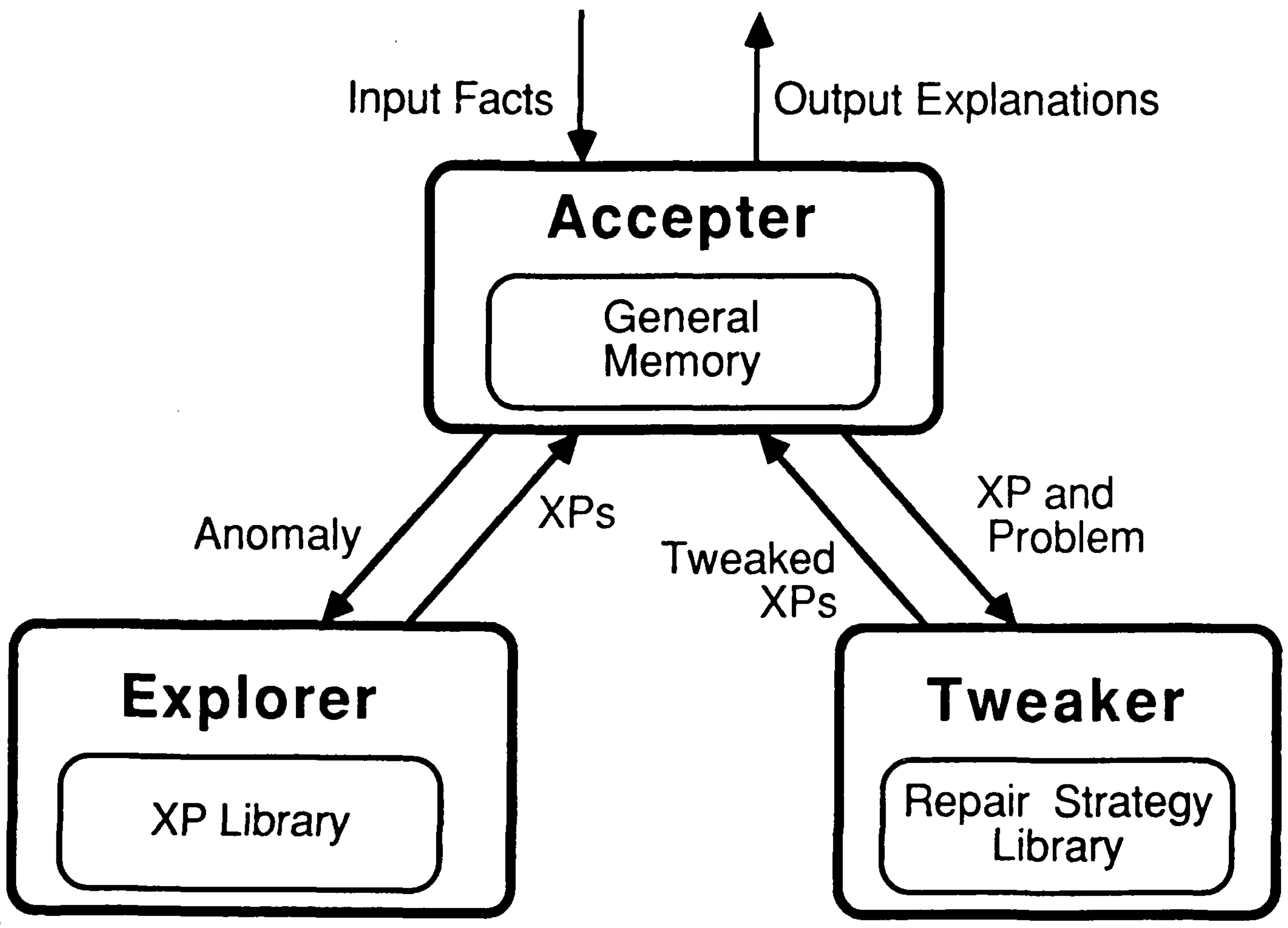
SWALE's Explanations:

Swale had a congenital heart defect.
The exertion of running in horse races strained his heart and brought out the latent defect.
He had a heart attack and died.

Swale's owner was giving him drugs to improve his performance.
He accidentally gave him an overdose.
The overdose killed him.

Swale was thinking about his future on the stud farm.
The excitement caused a heart attack.

SWALE Module Interconnection



Using Memory to Explain

- Reuse Old Processing Effort
- Adapt Knowledge to New Situations
- More Flexible than Pattern Instantiation
- More Efficient than Building Chains
- Use Explanation Patterns (XPs):

  Frozen Explanatory Inference Chains

An Input Story:

Swale was a successful three-year-old racehorse.

Swale won the Belmont Stakes.

Swale died a week later.

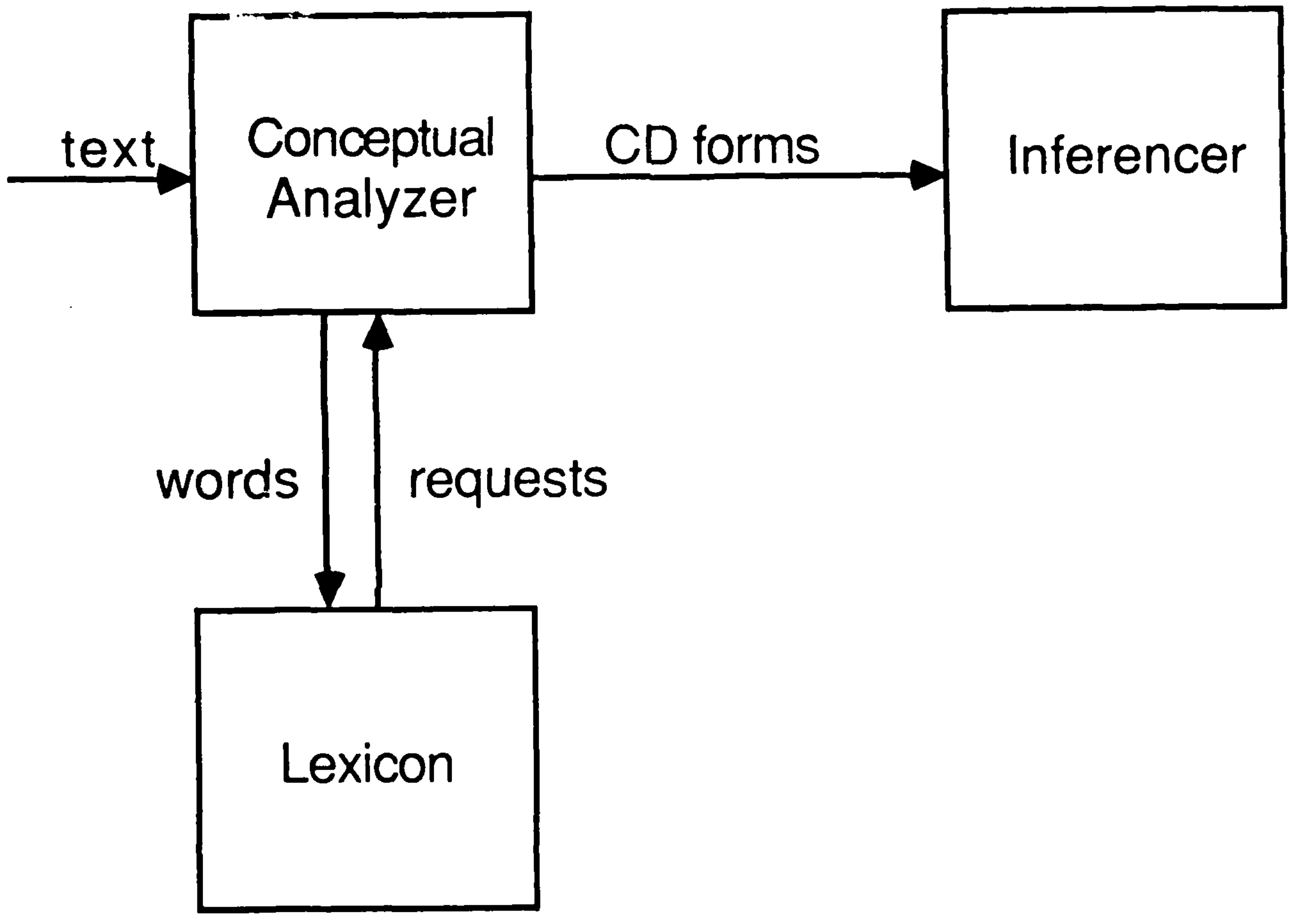
An Explanation Pattern

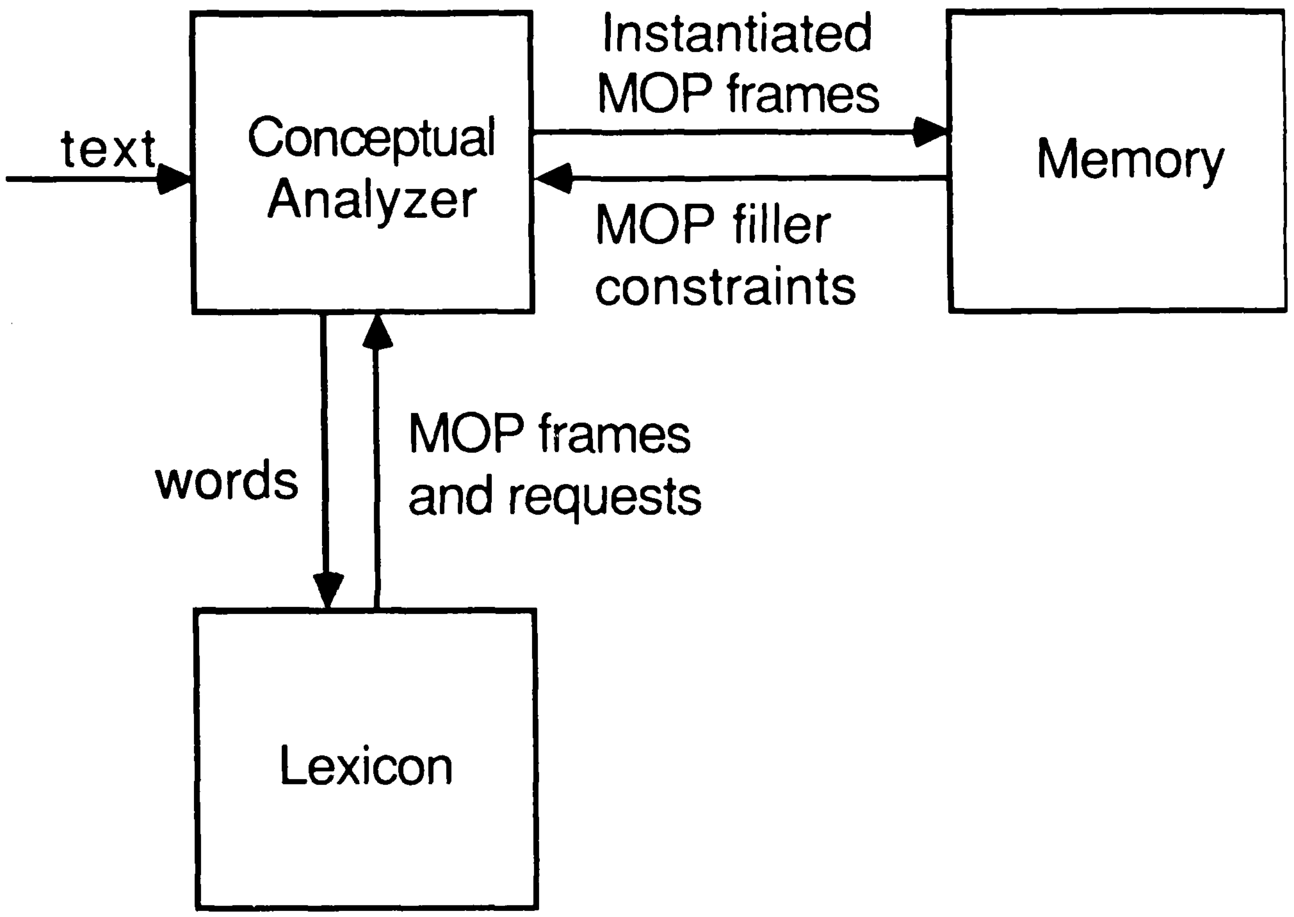
- Entry conditions : When is it applicable
- Relevance conditions: When is it relevant
  or a candidate for modification.
- Causal Chain

DIRECT MEMORY ACCESS PARSING

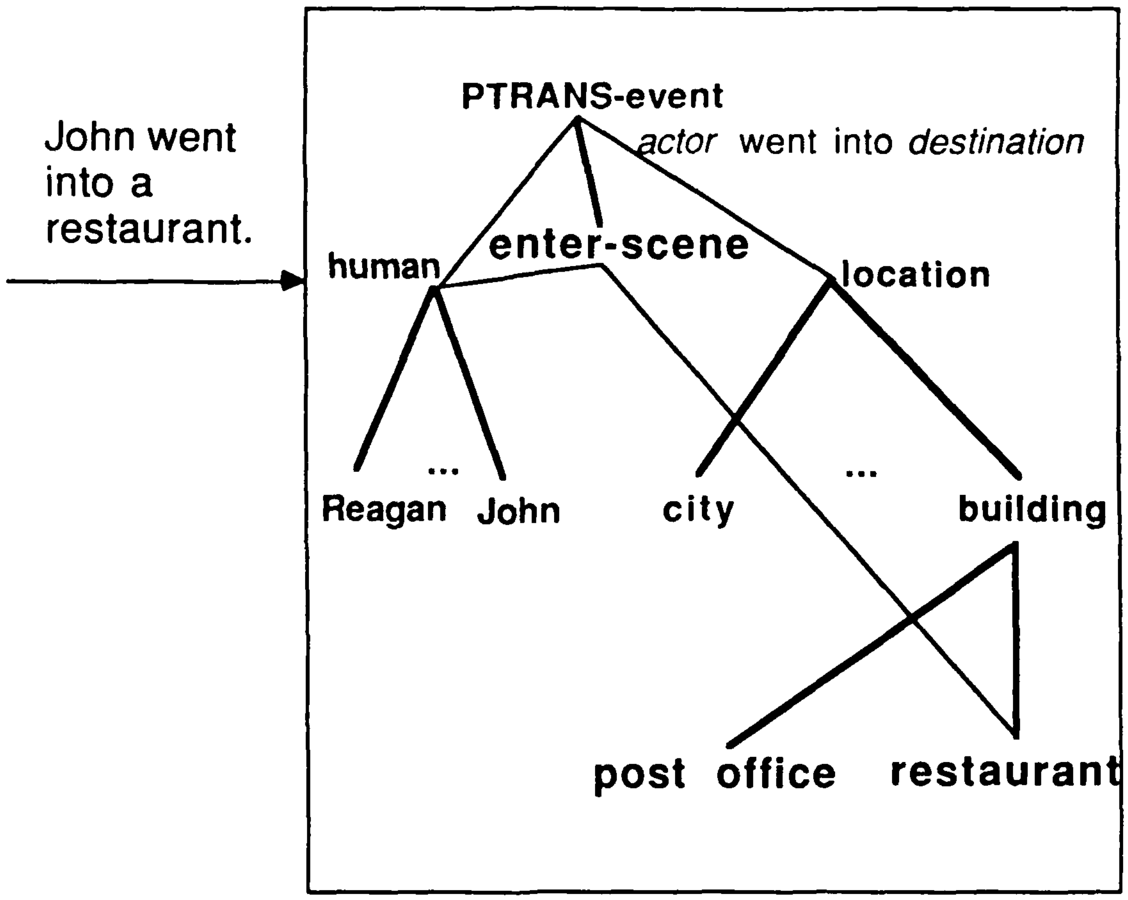
- only memory structure instantiation, no other form construction
- lexically-driven memory search process
- lexicon stored attached directly to concepts in memory

Parsing is treated as "recognize and record" rather than "build and store." A new text is either recognized as saying the same thing as an old text, or as being a specialization of a generalized text analysis. In the extreme, a whole article may be parsed as "same as article-25," if nothing new is said.


text
Conceptual Analyzer
CD forms
Inferencer
words
requests
Lexicon


text
Conceptual Analyzer
Instantiated MOP frames
MOP filler constraints
Memory
words
MOP frames and requests
Lexicon

MEMORY SEARCH PROCESSES



Direct Memory Access Parsing

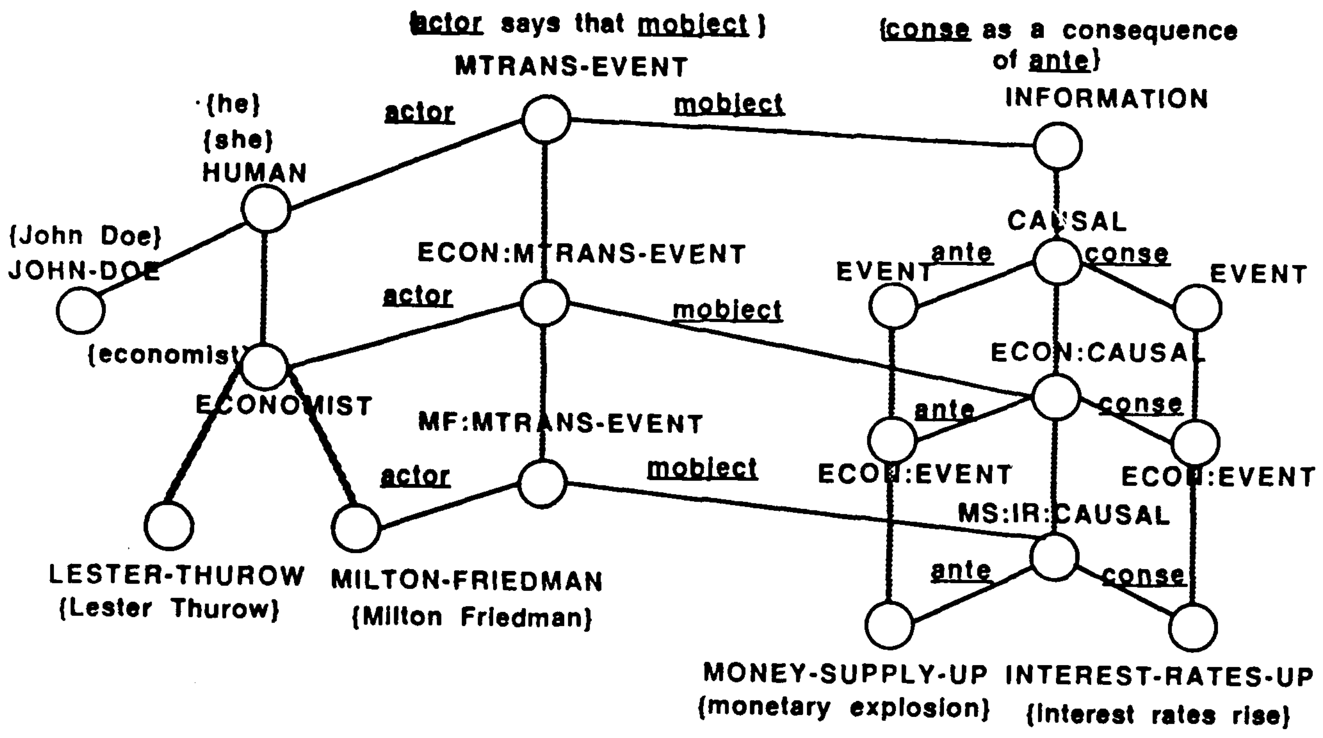
Memory Organization Packets (MOPs)

- Abstraction hierarchy (IS-A)
- Packaging hierarchy (PART-OF)

MOPs have zero or more concept sequences attached

- MTRANS-EVENT {actor says that mobject}
- MILTON-FRIEDMAN {Milton Friedman}

Concept lexicon consists simply of pointers from words, concepts, etc., to concept sequences.



{actor says that mobject}
MTRANS-EVENT
{conse as a consequence of ante}
INFORMATION
{he}
{she}
HUMAN
actor
mobject
{John Doe}
JOHN-DOE
CAUSAL
ante
conse
EVENT
EVENT
ECON:MTRANS-EVENT
actor
mobject
{economist}
ECONOMIST
ECON:CAUSAL
ante
conse
MF:MTRANS-EVENT
actor
mobject
ECON:EVENT
ECON:EVENT
MS:IR:CAUSAL
ante
conse
LESTER-THUROW
{Lester Thurow}
MILTON-FRIEDMAN
{Milton Friedman}
MONEY-SUPPLY-UP
{monetary explosion}
INTEREST-RATES-UP
{interest rates rise}

Markers

Two kinds of markers

- Activation markers
  - ◊ placed on words as they are read
  - ◊ placed on MOPs with fully activated concept sequences
  - ◊ passed up the abstraction hierarchy
- Prediction markers
  - ◊ placed on concept sequences indexed (in the concept lexicon) by activated structures
  - ◊ passed to concept sequence elements

Publications

Articles

Bain, W. A Case-based Reasoning System for Subjective Assessment. In Proceedings of the *Fifth National Conference on Artificial Intelligence.* Philadelphia, PA, August, 1986.

Kass, A. Modifying Explanations to Understand Stories. In Proceedings of the *Eighth Annual Conference of the Cognitive Science Society.* Amherst, MA, August, 1986.

Kass, A., Leake, D., and Owens, C. SWALE, A Program that Explains. In *Explanation Patterns: Understanding Mechanically and Creatively,* by R. C. Schank, In Press.

Martin, C., and Riesbeck, C. Uniform Parsing and Inferencing for Learning. In Proceedings of the *Fifth National Conference on Artificial Intelligence.* Philadelphia, PA, August, 1986.

Owens, C., and Leake, D. Organizing Memory for Explanation. In Proceedings of the *Eighth Annual Conference of the Cognitive Science Society.* Amherst, MA, August, 1986.

Riesbeck, C. Direct Memory Access Parsing. In proceedings of the *Second Annual Workshop on Theoretical Issues in Conceptual Information Processing.* New Haven, CT, May, 1985.

Riesbeck, C. Parsing, Expectation-Driven. In S. Shapiro (ed.) *Encyclopedia of Artificial Intelligence.* Forthcoming.

Riesbeck, C. From Conceptual Analyzer to Direct Memory Access Parsing: An Overview. In N. Sharkey (ed.) *Advances in Cognitive Science.* Forthcoming.

Riesbeck, C., and Martin, C. Direct Memory Access Parsing. In Kolodner and Riesbeck (eds.), *Experience, Memory, and Reasoning.* Lawrence Erlbaum Associates, Hillsdale, NJ. In press.

Riesbeck, C., and Martin, C. Towards Completely Integrated Parsing and Inferencing. In Proceedings of the *Eighth Annual Conference of the Cognitive Science Society.* Amherst, MA, August, 1986.

Schank, R., Collins, G. and Hunter, L. On the Failure of Inductive Category Formation as a Model of Learning. *Behavioral and Brain Sciences,* in press.

Schank, R., and Owens, C. Understanding by Explaining Expectation Failures. To appear in *Communication Failures In Discourse,* Edited by Ronan Reilly, North-Holland, In Press.

Schank, R., and Leake, D. Computer Understanding and Creativity. Invited paper for 1986 Conference of the International Federation for Information Processing. Dublin, Ireland.

Technical Reports

Schank, R. and Riesbeck, C. *Explanation: A Second Pass.* Technical Report 384, Yale Department of Computer Science. August, 1985.

Schank, R. and Riesbeck, C. *Questions and Thought.* Technical Report 385 Yale Department of Computer Science. August, 1985.

Books

Charniak, E., Riesbeck, C., McDermott, D., and Meehan, J. *Artificial Intelligence Programming. Second Edition.* Lawrence Erlbaum Associates, Hillsdale, NJ. In press.

Riesbeck, C. and Kolodner, J. (editors). *Experience, Memory, and Reasoning.* Lawrence Erlbaum Associates, Hillsdale, NJ. In press.

Schank, R. *Explanation Patterns: Understanding Mechanically and Creatively.* Lawrence Erlbaum Associates, Hillsdale, NJ. 1986.

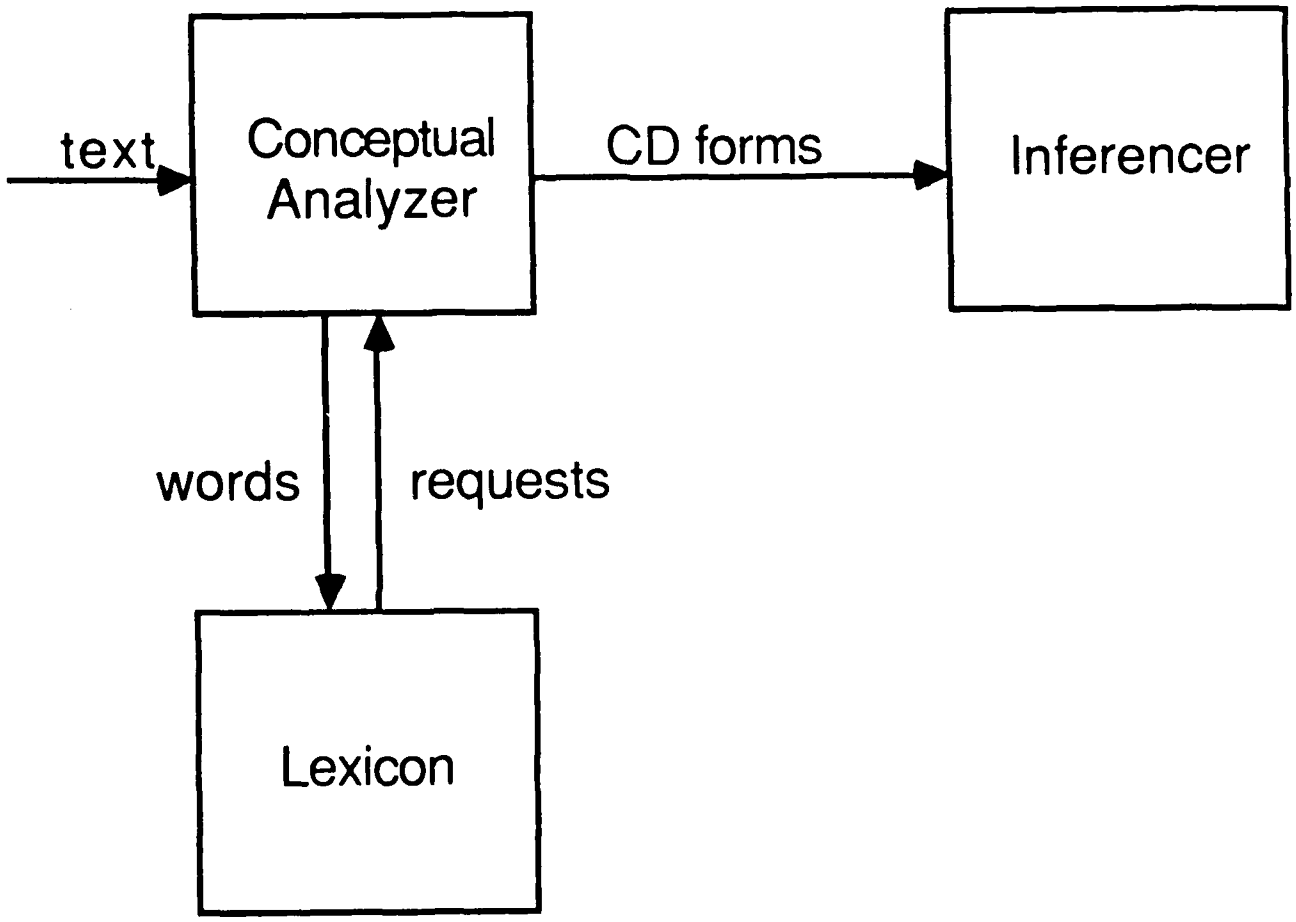
Schank, R. and Riesbeck, C. *Inside Computer Learning*, Lawrence Erlbaum Associates, Hillsdale, NJ. In preparation.

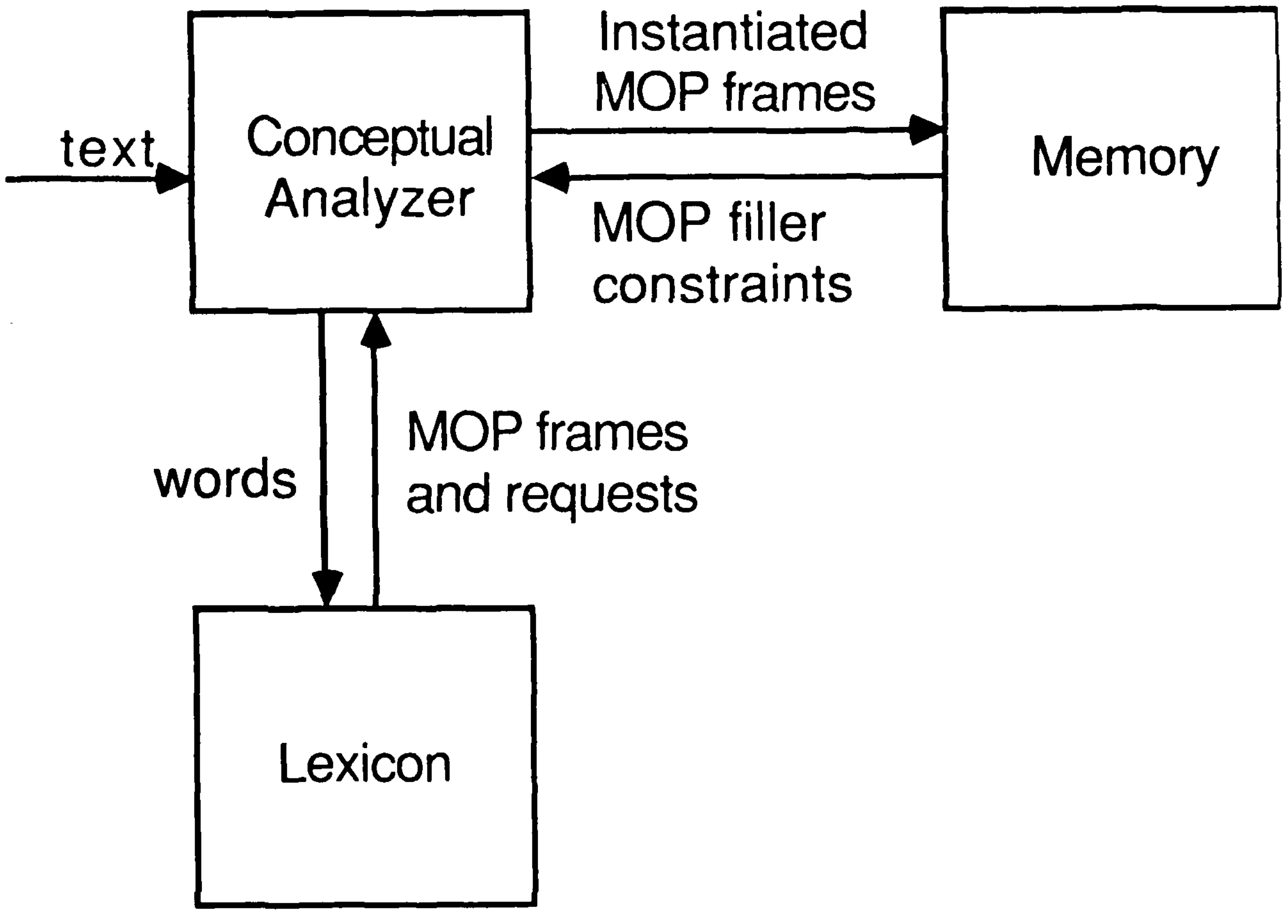
Slade, S. *The T Programming Language: A Dialect of LISP.* Prentice-Hall, Englewood Cliffs, NJ. In press.

Ph.D. Dissertations

Bain, William. *Case-based Reasoning: A Computer Model of Subjective Assessment.*

Hammond, Kristian. *Case-based Planning: Viewing planning as a memory task*


text
Conceptual Analyzer
CD forms
Inferencer
words
requests
Lexicon


Instantiated MOP frames
text
Conceptual Analyzer
Memory
MOP filler constraints
words
MOP frames and requests
Lexicon